Vers une approche orientée aspect d'ingénierie des besoins dans les organisations multi-entreprises

Mohammed Amroune

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par : *l'Université Toulouse 2 Jean Jaurès (UT2 Jean Jaurès)*

Présentée et soutenue le *24 Octobre 2014* **par :**

MOHAMMED AMROUNE

VERS UNE APPROCHE ORIENTÉE ASPECT D'INGÉNIERIE DES BESOINS DANS LES ORGANISATIONS MULTI ENTREPRISES

JURY

MAHMOUD BOUFAIDA	Professeur,Université de Constantine2	Président du Jury
PIERRE JEAN CHARREL	Professeur,Université de Toulouse	Directeur de thèse
NACEREDDINE ZAROUR	Professeur,Université de Constantine2	Directeur de thèse
JEAN MICHEL INGLEBERT	Maître de Conférences,Université de Toulouse	Co-Directeur de thèse
LIONEL SEINTURIER	Professeur,Université de Lille	Rapporteur
HASSINA SERIDI	Professeur,Université de Annaba,Algérie	Rapporteur
JEAN MICHEL BRUEL	Professeur,Université de Toulouse	Examinateur

École doctorale et spécialité :
MITT : Domaine STIC : Sureté de logiciel et calcul de haute performance

Unité de Recherche :
Institut de Recherche en Informatique de Toulouse (IRIT)

Directeur(s) de Thèse :
Pierre Jean CHARREL et *Nacereddine ZAROUR*

Rapporteurs :
Lionel Seinturier et *Hassina Seridi*

Mohamed AMROUNE

Vers une Approche Orientée Aspect d'Ingénierie des Besoins dans les Organisations multi Entreprises

Directeurs de thèse :
Nacereddine Zarour, Pr Université de Constantine II
Pierre Jean Charrel , Pr Université de Toulouse II

Résumé

Le système d'information coopératif (SIC) est un élément central dans le domaine de la coopération interentreprises. Son développement nécessite une attention particulière afin de prendre en considération tous les problèmes émergeants, surtout celui des préoccupations transversales qui posent des difficultés pour la compréhension, la maintenance, l'évolution, et la réutilisation des systèmes développés. Dans les approches usuelles de développement, la construction de ce genre de système part de zéro et nécessite de tout reconstruire à chaque fois. Nos travaux de recherche dans cette thèse examinent comment une approche orientée aspect appliquée de la phase de recueil des exigences à la phase de conception peut être proposée comme un outil permettant de développer des SICs à partir de systèmes d'information (SIs) préalablement existants produits à l'occasion de développements antérieurs. L'utilisation du paradigme Aspect dans cette approche tente de réutiliser des artéfacts des SIs existants afin de développer le futur SIC supportant la coopération interentreprises.

Mots Clés : Système d'information coopératif, exigences, réutilisation, Aspect, composition.

Institut de Recherche en Informatique de Toulouse - UMR 5505

Mohamed Amroune

Vers une Approche Orientée Aspect d'Ingénierie des Besoins dans les Organisations multi Entreprises

Supervisor :
Pr Nacereddine Zarour , Université de Constantine,
Pr Pierre Jean Charrel, Université de Toulouse II

Abstract

It is often difficult for a single Information System (IS) to accomplish complex requirements. One solution is to combine many different ISs and make them collaborate to realize this task. Information systems composition is an active ongoing area of research in the field of information systems. The result of IS composition produces one type of a so called Cooperative Information System (CIS). Its development requires a particular attention to process all emerging problems, especially the crosscutting concerns that pose difficulties to understand, maintain and reuse such cooperative systems. Moreover, the aspect paradigm is presented as a promising avenue for reusability. Thus, we argue that it is interesting to propose an aspect approach to build a new system in order to accomplish complex tasks based on the reuse of system's artefacts previously developed. According to our best of knowledge few works have tackled this question. In this thesis, we present an aspect-oriented approach called AspeCiS, applied from the requirements engineering phase until the design phase, in order to develop a CIS from existing ISs by using their artifacts such as requirements, architectures and design. Therefore, this approach is opposed to conventional development ones in which the construction of a new system starts from nothing and needs reinventing everything every time.

Keywords : Cooperative information system, requirements, aspect, reuse, composition.

Institut de Recherche en Informatique de Toulouse - UMR 5505
Université Paul Sabatier, 118 route de Narbonne, 31062 TOULOUSE cedex 4

Je dédie cette thèse
à la mémoire de mon cher père
à ma chère mère
à ma chère femme
à mes enfants (Oussama, Intissar et Ahlem)
à mes frères et à ma chère Sœur.

Remerciements

Tout d'abord, je tiens à exprimer mes plus vifs remerciements et ma gratitude à mes deux directeurs de thèse, Monsieur *Nacer eddine Zarour*, professeur de l'université de Constantine II, et Monsieur *Pierre-Jean Charrel*, professeur de l'université de Toulouse II, qui m'ont accueilli dans leurs laboratoires, le LIRE de Constantine et l'IRIT de Toulouse, pour leur encadrement continu et pour les remarques constructives qu'ils m'ont fournies. Je les remercie également pour la confiance qu'ils m'ont accordée et pour la grande liberté d'idées et de travail qu'ils m'ont donné. En dehors de ces apports scientifiques, je n'oublierai pas de les remercier pour leurs qualités humaines, leur hospitalité et leur soutien qui m'ont permis de mener à bien ma thèse de doctorat.

Je tiens à remercier très vivement mon encadrant Monsieur *Jean Michel Inglebert*, Docteur à l'Université de Toulouse II, pour son soutien, sa disponibilité, sa patience, la collaboration étroite dans laquelle nous avons travaillé et son aide qui m'ont permis de mener à bien ces travaux. Merci également pour ses relectures minutieuses de cette thèse.

Je tiens également à remercier les membres du jury qui m'ont fait l'honneur de bien vouloir évaluer mon travail : Je suis très sensible à l'honneur que m'a fait Monsieur Mahmoud Boufaida, professeur de l'université de Constantine II, en acceptant de présider mon jury.

Je remercie chaleureusement Monsieur Lionel Seinturier, professeur de l'université Lille et Madame Hassina Seridi, professeur de l'université d'Annaba, pour avoir accepté de rapporter cette thèse. Je leur exprime toute ma reconnaissance pour l'intérêt porté à ce travail. Leurs remarques apporteront sûrement matière à ma réflexion.

Je remercie vivement Monsieur Jean Michel Bruel, professeur de l'université Toulouse, pour avoir accepté d'examiner cette thèse.

Je remercie les membres des laboratoire LIRE et IRIT que j'ai côtoyé durant la période de ma thèse et qui ont su rendre mon travail agréable à travers leur simple présence et l'ambiance qu'ils ont su créer particulièrement *Hakim Bendjenna, Karim*

Zarour, Younes Lakhrissi, Adil Anwar, Toufik Dkaki, Rahma Bouaziz, Mahmoud Elhamlaoui, Adel Ziani, Samia Iltache, et Soumia Bendekir.

Je remercie du plus profond de mon cœur ma chère femme pour son soutien, sa patience et son amour. Je suis très reconnaissant pour les sacrifices qu'elle a fait pendant mes longues années d'études et d'absence et je demande pardon à mes enfants (*Oussama, Intissar et la petite Ahlem*) de mes absences répétées et d'avoir utilisé une partie du temps qui leur était dû, pour préparer ma thèse.

Table des matières

Liste des figures

Liste des tableaux

Introduction générale

Contexte du travail

Les entreprises de nos jours évoluent dans des environnements caractérisés par l'intensification de la concurrence, les changements touchant par exemple les demandes clients ou la performance des communications. Si ces entreprises veulent s'intègrer dans cette nouvelle configuration, et souhaitent atteindre de nouveaux objectifs, elles doivent avoir toutes les compétences nécessaires. Mais dans le cas où une entreprise ne dispose pas de toutes les compétences nécessaires et de toutes les ressources (humaines, technologiques et financières) suffisantes pour faire fâce aux défis de ce nouveau contexte, la coopération interentreprises est une des solutions possibles. La coopération interentreprises est une forme de relation entre les entreprises qui leur permet, en agissant ensemble, d'atteindre un objectif autrement irréalisable.

Plusieurs formes de coopérations interentreprises existent (entreprise virtuelle, réseau d'entreprises, ...) [Zarour, 2004], [Bouzguenda, 2005], [Norta and Grefen, 2006], [Grefen et al., 2009]. Ces formes d'organisation ont un impact important sur les systèmes d'information. En particulier, la flexibilité et l'ouverture vers l'environnement deviennent des enjeux majeurs dans la conception et la réutilisation de ces systèmes. Les systèmes d'information pour chaque type de coopération doivent être en mesure de supporter ces enjeux.

Nos travaux de recherche s'intéressent au développement d'un système d'information coopératif (SIC) supportant la coopération interentreprise. En particulier le développement d'une approche orientée aspect, d'élicitation des exigences. On considère que le système d'information est un élément central de cette problématique de coopération interentreprise. Ce système d'information ne se développe pas à partir de rien, mais on envisage de ne considérer que des systèmes construits à partir de systèmes (pré-)existants. De tels besoins sont rencontrés dans les plates-formes d'intégration de différents SI d'entreprises, dans les systèmes basés sur l'intégration de données (web ou autre), dans les systèmes construits à partir de sous services (web services) etc.

La réutilisation des systèmes, permet un développement de nouveaux systèmes d'information en réutilisant les artefacts des systèmes existants. Ces idées ne sont pas nouvelles [Parnas, 1972], [Frakes and Sadahiro, 1994], [Poulin, 1995], et tentent de maitriser la complexité et de réduire le coût et le temps de développement. Le paradigme objet est certainement l'un des exemples les plus répandus d'usage de la réutilisation. La réutilisation se définit comme une approche de développement de systèmes selon laquelle il est possible de construire un système à partir d'artefacts existants produits à l'occasion de développements antérieurs. Cette approche s'oppose aux approches usuelles de développement dans lesquelles la construction d'un nouveau système part de zéro et nécessite de tout reconstruire à chaque fois.

Toutefois, les différentes préoccupations que les systèmes doivent prendre en compte se retrouvent bien souvent enfouies ou éparpillées [Kiczales et al., 1997], [Awais et al., 2003], [Awais et al., 2002]. C'est de ce constat que sont nées les approches de programmation par aspects [Kiczales et al., 1997]. De ce fait, l'approche envisagée dans cette thèse, pour le développement d'un système d'information coopératif, devra prendre en considération ce problème des préoccupations transverses.

Problématique

Les méthodes de développement logiciel définissent toutes les phases de l'ingénierie des exigences (IE) jusqu'à l'implémentation. Les travaux de recherche qui se focalisent sur les premières phases du cycle de vie, sont de nos jours un domaine d'actualité et de recherche. Ces travaux s'intègrent dans le domaine de l'IE, cherchant à développer des approches, des techniques et des outils pour bien mener cette phase. Il est évident aujourd'hui, que les échecs dans la mise en œuvre et l'utilisation des projets SI sont dûs principalement aux mauvaises compréhensions des exigences auxquelles ces systèmes tentent de satisfaire. Ce constat d'échec est mentionné dans plusieurs travaux et rapports de recherche [Standish 2003, Standish 2007, ...]. A ces problèmes s'ajoute le problème des préoccupations transversales, qui posent problème pour la compréhension, la maintenance, l'évolution, et la réutilisation des systèmes développés.

L'utilisation du paradigme Aspect dans la phase d'implémentation, qui apporte des solutions aux difficultés du paradigme de l'objet, en particulier aux problèmes de dispersion et d'enchevêtrement du code, est aujourd'hui disponible pour un nombre important de langages de programmation orientés aspect (aspectJ, hyperJ, AspectC, JAC, etc.). Cependant l'utilisation du paradigme aspect, lors des premières phases du cycle de développement pour faire face à la complexité des applications n'a pas atteint encore un stade de maturité. Beaucoup de travaux sont en cours notamment dans la phase d'IE orientée Aspect (Aspect Requirements Engineering : AORE).

Les travaux effectués dans le cadre de cette thèse tentent :

d'examiner comment une approche orientée aspect appliquée de la phase de recueil des exigences à la phase de conception peut être proposée comme un outil permettant de développer des systèmes d'information coopératifs à partir de systèmes préalablement existants ?

Une approche classique consisterait à analyser, concevoir puis développer le nouveau système par un ensemble de modules logiciels s'appuyant sur les sous systèmes composants.

L'approche envisagée dans ce travail consiste à analyser les exigences du nouveau système à développer, les considérer comme des aspects (précoces) et ensuite les tisser (propager) sur les systèmes existants (plus précisément sur leurs parties visibles et accessibles).

Le coté original de cette approche est que les exigences fonctionnelles du nouveau système sont appréhendées avec une technique qui habituellement ne concerne que les exigences non fonctionnelles (persistance, parallélisme, trace, etc).

Les interrogations essentielles que nous avons recensées en abordant ce travail sont :
- Quelle méthodologie d'analyse proposer pour capturer les nouvelles exigences sous forme d'aspects ?
- Quels sont les points de jonction pour tisser des aspects fonctionnels sur un système existant ? doit-on envisager un ensemble prédéfini de points de jonctions ? un langage de définition de points de jonctions ?
- Pourra-t-on proposer des types d'aspects fonctionnels ? y retrouvera-t-on les modèles correspondants ?
- Dans quelles conditions, un tissage fonctionnel est-il réapplicable/réutilisable ?

Cette approche offre plusieurs avantages tels que :

(i) Elle permet de ne pas mélanger dans le système produit les fonctionnalités pré-existantes des nouvelles fonctionnalités.

(ii) Elle fournit un degré de *réutilisation fonctionnelle* qui pourrait permettre de *tisser* à nouveau ces mêmes fonctionnalités sur d'autres systèmes pré-existants (réellement différents ou les mêmes dans une autre version etc).

Objectifs et démarche

L'idée générale de notre contribution est que le processus d'élicitation des exigences d'un système d'information coopératif inter-organisationnel peut être construit en réutilisant les systèmes existants par une approche aspect. En particulier, l'approche proposée est destinée à établir une démarche d'élicitation des exigences en se basant sur le principe

de la séparation des préoccupations. Les exigences une fois définies, sont modélisées en examinant cette notion de séparation, au niveau modèle. Puis un processus de tissage est appliqué aux modèles des exigences décomposées en exigences de bases et exigences transversales.

Structure du mémoire

La structure en cinq chapitres de ce mémoire reflète notre démarche de travail.
Chapitre I : **La séparation des préoccupations et le Paradigme Aspect**
Dans ce chapitre, nous présentons les concepts de base de la séparation des préoccupations, ainsi que le concept des préoccupations transversales. Le paradigme Aspect est présenté avec plus de détails, en présentant les limites du paradigme objet et le passage de l'objet aux Aspects. AspectJ est présenté comme un exemple des familles des langages orientées aspect.
Chapitre II : **L'ingénierie des exigences orientées Aspect**
Dans ce chapitre, nous introduisons le domaine de l'ingénierie des exigences. En particulier nous présentons quelques définitions, objectifs ainsi que le processus d'ingénierie des exigences. Une partie de ce chapitre est consacrée à des exemples d'approches d'IE dites classiques (non orientées aspect) ainsi que des exemples d'approches orientées aspect. L'objectif de ce chapitre est de fournir une base théorique pour notre recherche en analysant les approches existantes d'ingénierie des exigences orientées Aspects.
Chapitre III : **La composition de modèles dans l'Ingénierie Dirigée par les Modèles**
Etant donné que l'approche proposée dans cette thèse est une approche dirigée par les modèles, le chapitre présente l'ingénierie dirigée par modèles (IDM) de façon générale. Des définitions des principaux concepts sont présentées. L'accent est mis sur la composition des modèles dans l'approche IDM, et en particulier le concept de tissage de modèle. Nous présentons dans ce chapitre aussi quelques approches de composition et de tissage de modèles.
Chapitre IV : **AspeCiS : une approche orientée aspect pour l'élicitation et la modélisation des exigences pour un système d'information coopératif**
Ce chapitre présente l'approche proposée dans ce travail, à savoir une approche orientée aspect dirigée par les modèles, pour l'éliciatation et la modélisation des exigences dans les environnements multi entreprises.
Chapitre V : **Etude de cas.**
Dans ce chapitre nous montrons l'utilisabilité et l'efficacité d'AspeCiS à travers une étude de cas dans le domaine de l'enseignement supérieur. Nous présentons le domaine d'application de notre cas d'étude puis l'implémentation du métamodèle de tissage d'AspeCiS.
Ce mémoire de thèse se termine par le bilan des différents travaux réalisés dans la thèse

et les travaux que nous envisageons dans le futur.

Première partie

Etat de l'Art

1 La séparation des préoccupations et le Paradigme Aspect

Diviser chacune des difficultés en autant de parcelles qu'il se pourrait, et qu'il serait requis pour les mieux résoudre (Descartes).

1.1 Introduction

Parmi les objectifs les plus recherchés en génie logiciel se trouvent la modularisation, la lisibilité et la compréhension des logiciels, la possibilité de les réutiliser, et de les faire évoluer facilement et de manière fiable. Afin d'atteindre de tels objectifs et favoriser une ingénierie des systèmes d'information (SIs) de bonne qualité, de nombreux modèles (Objets, composants, agents, etc.) et méthodologies de développement (Merise, OMT (Object Modeling Technique), UP (Unified Process), etc.) ont été adoptés ces dernières années.

Les approches les plus utilisées sont certainement, les approches Objet et Composant. Cependant, et malgré les avantages de ces modèles et méthodes, ils ont aussi leur lot d'inconvénients et présentent un certain nombre de lacunes et problèmes qui les rendent, aujourd'hui, insuffisants afin de prendre en compte la complexité croissante des nouvelles applications. Ainsi, devant la complexité des SIs et leurs besoins d'évolution qui rend leur développement plus difficile, plus couteux et moins fiable, l'ingénierie des SIs se trouve dans l'obligation d'explorer de nouvelles problématiques d'étude : notamment, l'ingénierie de nouvelles techniques et méthodes de développement afin de faciliter l'évolution et favoriser la réutilisation des systèmes produits. Cette pratique se traduit aujourd'hui par l'existence d'approches de développement basées sur la notion d'Aspect, l'Ingénierie Dirigée par les Modèles (IDM ou MDA pour Model Driven Architecture), ou les méthodes agiles.

Pour faire face à la complexité du développement des SIs, la séparation des préoccupations, lors du cycle de développement des applications, s'est avérée être une solution prometteuse. La section suivante décrit ce principe.

1.2 La séparation des préoccupations

La séparation des préoccupations (SoC, Separation of Concerns) est un concept présent depuis de nombreuses années dans l'ingénierie des logiciels. Les différentes préoccupations des concepteurs apparaissent comme les motivations premières pour organiser et décomposer une application en un ensemble d'éléments compréhensibles et facilement manipulables. La séparation en préoccupations apparaît dans les différentes étapes du cycle de vie du logiciel et elles sont donc de différents ordres. Il peut s'agir de préoccupations d'ordre fonctionnel (séparations des fonctions de l'application), technique (séparation des propriétés du logiciel système), ou encore liées aux rôles des acteurs du processus logiciel (séparation des actions de manipulation du logiciel). Par ces séparations, le logiciel n'est plus abordé dans sa globalité, mais par parties.

Une préoccupation est définie comme étant *l'abstraction d'un but particulier ou une unité modulaire d'intérêt* [Hachani, 2006]. Un système typique contient essentiellement deux catégories de préoccupations : des préoccupations fonctionnelles et des préoccupations non fonctionnelles. Les préoccupations apparaissent dans les différentes phases du cycle de développement. Nous pouvons distinguer par exemple : des préoccupations liées aux données, à leur persistance et à leur contrôle d'accès, ou encore, des préoccupations architecturales telles que l'interopérabilité entre applications ou des préoccupations de gestion organisationnelle. A titre d'illustration, dans un système de gestion de cartes de crédit, le processus de paiement est une préoccupation fonctionnelle ; l'intégrité de la transaction de paiement et la gestion de l'authentification lors de la connexion sont des préoccupations non fonctionnelles.

L'objectif principal de la séparation des préoccupations est de représenter de manière abstraite et explicite, à tous les niveaux de développement, les différentes préoccupations d'un système, et de les adresser indépendamment tout en identifiant leurs interrelations afin de les composer . La séparation des préoccupations est un principe clé pour la réduction de la complexité du développement de grands systèmes. Si on arrive à bien isoler et localiser le code relatif à toute préoccupation particulière d'un système, ceci présente plusieurs avantages. Il devient ainsi plus facile de comprendre comment la préoccupation est adressée dans le code qui devient facile à étendre, à modifier et à réutiliser.

En adoptant l'idée classique de la modularité utilisée par l'ensemble des approches de développement logiciel, le principe de séparation des préoccupations identifie de manière séparée l'ensemble des préoccupations d'un système logiciel. Ces préoccupations

sont adressées de manière relativement indépendante lors des phases de l'Ingénierie des Exigences (IE), de la conception et aussi lors la phase d'implémentation. Ce principe est devenu essentiel au développement d'une application logicielle, car il améliore la lisibilité et la flexibilité, et permet l'évolution, l'adaptation et la réutilisation de toute ou partie de l'application. Cette approche offre plusieurs avantages qui seront présentés dans la section suivante.

1.2.1 Les préoccupations transversales

Le principe de séparation des préoccupations exprime une bonne modularisation et une bonne qualité du code, il n'indique pas cependant comment arriver par un processus précis de développement à implémenter des préoccupations. Plusieurs modèles et langages de programmation offrent, en ce sens, différents mécanismes et concepts qui permettent une meilleure organisation des programmes selon ce principe. Il s'agit en général d'organiser les programmes en unités modulaires séparées (procédures, fonctions, objets, etc.) concernant chacune une préoccupation particulière. Cependant, comme le note Kiczales et al. [Kiczales et al., 1997], les méthodes classiques telles que les méthodes orientées objet traitent essentiellement de la séparation et de la composition de préoccupations fonctionnelles. Elles ne garantissent pas en revanche, une représentation modulaire et séparée des préoccupations, qui affectent à la fois plus d'une unité modulaire fonctionnelle. Le code de telles préoccupations, dites transversales (Crosscutting Concerns), se trouve ainsi dispersé et enchevêtré dans l'ensemble du code des préoccupations fonctionnelles déjà implémentées. Ceci soulève plusieurs problèmes préjudiciables à la compréhension, la maintenance, l'évolution et la réutilisations du code relatif à ces préoccupations transversales, mais aussi de celui relatif à l'application.

Il existe différents types de préoccupations transversales, dont les principales familles sont les suivantes [Kiczales et al., 1997] :

- préoccupations liées aux données (persistance de données, contrôle d'accès, etc.).
- préoccupations de logging/tracing, de gestion d'exceptions, de debugging et de tests, etc.
- préoccupations de gestion organisationnelle, workflow, etc.
- préoccupations techniques et architecturales, telles que la configuration, la distribution, la synchronisation, l'interopérabilité entre applications hétérogènes, la sécurité, la gestion de performance, etc.

1.2.1.1 Exemple de préoccupations transversales

Le processus d'analyse d'une application informatique génère un ensemble de classes contenant des données et des traitements. Ces classes qui regroupent un ensemble d'ob-

jets peuvent dans certains cas être corrélées. Pour illustrer ce phénomène de corrélation, nous reprenons l'exemple des contraintes d'intégrités référentielles cité dans [Renaud et al., 2004].

Cet exemple contient deux classes à savoir la classe *Client* et la classe *Commande*. Avant de supprimer un *Client* il faut s ?assurer qu ?il n ?a aucune *Commande* non satisfaite dans la classe Commande. Cependant, la suppression de la *Commande* peut engendrer la perte des coordonnées du *Client* concerné. Une solution à ce problème, basée sur la modification de la méthode de suppression de la classe *Client*, avec la vérification au préalable de absence de *Commande* non satisfaites relevant du *Client* à supprimer. Cette solution, qui peut considérer bonne, présente cependant quelques inconvénients [Renaud et al., 2004] :

- la vérification des commandes non honorées ne fait pas partie de la logique de gestion d'un *Client*. La classe *Client* devrait gérer les fonctionnalités qui relèvent de sa propre logique.
- une classe n'est pas censée prendre connaissance de toutes les contraintes d'intégrité dictées par les autres classes de l'application.
- étant donnée que la classe *Client* implémente des fonctionnalités liées à d'autres classes, ceci limite les possibilités de sa réutilisation indépendamment des classes liées.

Vu les raisons citées précédemment, il est claire que la classe *Client* n'est donc pas le meilleur emplacement pour implémenter cette contrainte d'intégrité référentielle. dans l'absence de meilleure solution, plusieurs programmes objet implémentent cette contrainte de cette façon. En plus, la classe *Commande* n'est pas aussi adaptée à l'implémentation de cette contrainte : si nous cherchons à supprimer un *Client*, il n'y a aucune raison pour vérifier si la classe *Commande* le permet.

Cette contrainte n'est pas propre ni de la classe *Client* ni de la classe *Commande* mais elle est transversale aux deux classes. Les fonctionnalités transversales posent problème pour le découpage en classes qui a pour but de rendre les classes indépendantes entre elles. La Programmation Orientée Objet (POO) ne possède aucune solution pour prendre en compte ces fonctionnalités transverses. En POO, l'implémentation d'une méthode est localisée dans une classe, et son invocation, ou utilisation se trouve dispersée. Cette dispersion est un frein à la maintenance et à l'évolution des applications orientées objet.

1.2.2 L'impact de la séparation des préoccupations

L'objectif principal de la séparation des préoccupations est la séparation entre les fonctionnalités qu'un logiciel accomplit et les préoccupations qui contraignent le déroulement de ces fonctionnalités. Cette approche offre plusieurs avantages [Meslati, 2005].

- **Elimination des dépendances** : les dépendances sont en étroite relation avec l'enchevêtrement. La séparation des préoccupations est un atout important pour éliminer l'enchevêtrement des préoccupations avec les fonctionnalités et les autres préoccupations. Cependant, les changements qui peuvent toucher certaines préoccupations auront un effet réduit et bien cerné sur les autres. Par exemple, le fait de définir, dans un aspect le code de la synchronisation cela permet de supprimer des types de dépendances telles que les dépendances de définition et d'héritage.
- **Séparation des services communs** : une application contient aussi des services qui sont constitués par des préoccupations. La séparation des préoccupations permet alors l'évolution et la maintenance de ces services en éliminant l'enchevêtrement du code. Cependant, un service peut subir des changements sans modifier le reste de l'application. Par exemple, si on veut opérer un changement sur la stratégie de sécurité ou de synchronisation de l'application, il suffit d'effectuer ces changements au niveau des préoccupations.
- **Maintien de la séparation** : la séparation des préoccupations aura un meilleur impact si elle est maintenue sur toutes les phases de développement, c'est-à-dire de la phase d'analyse jusqu'à la phase d'implémentation. Le code exécutable de l'application est généré par le tissage des codes des préoccupations, ce qui va engendrer l'enchevêtrement du code et détruit la séparation. Il est facile de modifier, maintenir et faire évoluer les applications où la séparation au niveau code est reflétée au niveau de la conception. Si l'évolution est opérée sur le code source et les niveaux qui lui sont supérieurs, on peut considérer que la séparation est persistante.
- **Doter le système de moyens d'adaptation dynamique** : deux types de tissage sont définis à savoir : le tissage statique qui s'effectue soit au niveau modèle soit au niveau code, et le tissage dynamique qui s'effectue au moment de l'exécution. Ce dernier offre la possibilité d'ajouter ou de retirer de manière dynamique des aspects sans conséquences sur le reste de l'application, comme le font certains tisseurs tels que (JAC : Java Aspect Component).

La séparation des préoccupations est utilisée dans des approches telles que la programmation par aspect, la composition des filtres, la programmation adaptative, la programmation générative, ou encore la programmation par sujet. Ces approches sont présentées dans les sections suivantes.

1.3 La séparation des préoccupations et les approches classiques : limites et problèmes

Nous discutons, dans cette section, les limites et problèmes des principales approches classiques bâties autour du paradigme Objet, vis-à-vis de la séparation, la modularisation

et l'intégration des préoccupations transversales.

1.3.1 La programmation procédurale, modulaire et par objets

La programmation procédurale ou fonctionnelle, met l'accent sur les fonctionnalités attendues du système concret. L'ensemble des constructions du programme partagent les données du système. La programmation procédurale atteint ses limites avec l'augmentation de la complexité des SIs. Ainsi, la réutilisation, l'évolution ainsi que la maintenance des applications à base de procédures et fonctions deviennent difficiles.

Les langages de la programmation modulaire, tels que, le langage Modula-2 [Wirth, 1985] et le langage ADA [Booch, 1981] utilisent les deux principes d'encapsulation et d'abstraction des données. Les procédures et/ou fonctions d'une application sont dépendantes selon ces deux principes. Les données manipulées sont regroupées et séparées dans un module qui définit un type abstrait. Plusieurs modules fonctionnels constituent, alors, un programme dans lequel la collaboration inter modules se fait par le biais des interfaces. Cependant, la décomposition des programmes en modules ne garantit pas l'application du principe de la séparation des préoccupations, à cause des différentes interactions qui se trouvent non encapsulées et dispersées.

La programmation par objets (Object Oriented Programming ou OOP) vise à apporter une solution aux problèmes posés par la programmation modulaire. Les langages à objets utilisent une structure en classes d'objets. Par le biais de cette structure, l'approche objet assure la réutilisation des programmes développés. Elle facilite également l'évolution et l'extension des classes de manière incrémentale. La programmation par objets est largement utilisée par un nombre important de langages de programmation tels que (Smalltalk [Goldberg and Robson, 1983], Eiffel [Meyer, 1992], [Jezequel, 1996], Java, etc.) et beaucoup d'outils de développement.

Les travaux de [Victor, 2004], [Sadou et al., 2005] mentionnent que la programmation par objets se trouve mal adaptée pour une représentation explicite, modulaire et séparée d'interactions complexes entre groupes d'objets, en plus, le niveau de granularité très bas de cette approche (ie : objet/classe) pose quelques problèmes [Meyer, 1997], [Villalobos, 2003]. Le découpage d'une application en terme de classes n'est pas en mesure d'assurer une réelle solution aux problèmes de dispersion et d'enchevêtrement du code d'une application.

1.3.2 La programmation réflexive

Les langages réflexifs, tels que Smalltalk [Goldberg and Robson, 1983], ObjVLisp [Cointe, 1987], Common Lisp Object System (CLOS) [Kiczales et al., 1994], CodA [McAffer, 1995], OpenC++ [Shigeru, 1995] et OpenJava [Michiaki et al., 2000], sont basés sur

des protocoles à MetaObjets (Meta-Object Protocols ou MOP) [Kiczales et al., 1991]. Le modèle d'un langage réflexif est décrit dans ce même langage. Sa modification, son adaptation peuvent se faire aussi par programme. Si plusieurs préoccupations transversales affectent une classe donnée, elles doivent être toutes définies dans la même métaclasse associée à la classe affectée. Ce qui augmente l'enchevêtrement du code et ne garantit pas la séparation entre préoccupations transversales. Autrement dit, le problème des différentes préoccupations qui se retrouvent enchevêtrées dans les métaclasses n'est pas traité par l'approche de la réflexivité. Cette approche ne permet pas aussi d'exprimer clairement le protocole de composition des différentes préoccupations qui se retrouvent enchevêtrées dans les métaclasses.

Le cadre extrêmement général de la réflexivité constitue sa faiblesse principale. En effet, la réflexivité n'est pas capable de traiter de façon rigoureuse le problème des préoccupations transversales. A cet effet, de nombreux efforts restent nécessaires pour prendre en charge ce problème qui freine la maintenance et la réutilisation.

1.3.3 La programmation générique

Le paramétrage de classes est le principe sur lequel est fondée la programmation générique. Le degré d'ouverture d'une classe définie dans un langage générique est dépendant de un ou plusieurs paramètres génériques formels, que peut avoir cette classe. Dans la littérature, plusieurs langages supportent la programmation générique, à titre d'exemple, on trouve : CLOS [Kiczales et al., 1994], C++ [Stroustrup, 1997], Eiffel [Meyer, 1992], et Java (a partir de la version 1.5). Les préoccupations liées à la définition de structures sont découplées de la spécification du ou des types de données concrets que détiennent ces structures dans les classes génériques. Il s'agit de la séparation de préoccupations de données.

D'autres techniques dérivées de la programmation générique comme : les Mixins [Bracha and Cook, 1990], et les Mixins Layers [Smaragdakis, 1998] permettent la séparation d'autres types de préoccupations transversales. Toutefois, la programmation générique et ses dérivés souffrent aussi de certaines limites, d'autant plus, qu'elles correspondent à des approches trop restrictives de séparation de préoccupations. La mise en en œuvre des Mixins lorsqu'elle est basée sur l'utilisation de l'héritage est aussi problématique, surtout dans le cas où le langage choisi pour le codage ne permet pas l'héritage multiple (tel que Java, par exemple).

1.3.4 La programmation à base de composants

La programmation à base de composants [Bosch et al., 2002] vise la réutilisation de composants fonctionnels, par une nouvelle approche de séparation de préoccupations

techniques (telles que la persistance, la synchronisation, la distribution, etc). En s'inspirant de l'assemblage des composants électroniques ou des composants mécaniques, les approches à base de composants permettent de concevoir et de développer des systèmes par assemblage de composants réutilisables.

Dans cette technique de programmation, la décomposition des applications en préoccupations relativement indépendantes est l'idée principale. Elle vise à adresser chaque préoccupation sans se référer aux autres à l'aide d'interfaces ou de contrats. Les préoccupations techniques transversales aux unités fonctionnelles (métiers) sont adressées séparément. Pour cela, cette technique offre de nombreux concepts tels que les notions d'interfaces requises et fournies ou les contrats, les intercepteurs, les conteneurs ou encore les événements. Ces moyens permettent aux composants de mieux séparer ces préoccupations techniques, ce qui offre une meilleure structure du programme, et permet donc de mieux cerner sa complexité [Brooks, 1995].

Cependant, ces approches ont aussi des problèmes de dispersion et d'enchevêtrement de code liés à certaines préoccupations transversales. En effet, les systèmes à base de composants utilisent les modèles objets comme support pour la spécification, la conception et l'implémentation des composants et leurs interactions [Garlan, 2000], [Smeda et al., 2005], [Medvidovic, 1999].

1.4 Synthèse des approches classiques

De l'étude précédente des principales approches classiques de développement, il en résulte que ces dernières ne sont pas en mesure d'assurer une séparation de tous les types de préoccupations.

- La décomposition de programmes en classes proposée par l'approche Objet souffre de plusieurs limites. Les préoccupations transversales restent dispersées dans plusieurs classes dans cette décomposition. Par ailleurs, la collaboration des groupes d'objets, l'invocation de méthodes ou la composition sont le seul moyen d'interconnecter différentes classes d'objets, ce qui rend difficile la séparation d'interactions complexes entre les groupes d'objets.
- La métaprogrammation comme moyen unique de séparation des préoccupations transversales reste restrictive et difficile à mettre en œuvre.
- Le paradigme de la programmation générique n'est pas capable d'assurer la séparation de tous les types de préoccupations.
- Les modèles à base de composants bien qu'ils permettent une mise en œuvre de manière transparente et flexible de certaines préoccupations techniques (telles que la gestion de la distribution, de la synchronisation, des transactions ou encore de la persistance), ils ne prennent en compte qu'un nombre limité de préoccupations.

En outre, les approches de développement classiques diffèrent par la manière d'appliquer la composition et l'adaptation des préoccupations (transversales ou non). Cependant, pour assurer la séparation d'un ou de plusieurs types de préoccupations transversales chaque approche possède ses propres solutions spécifiques. Ces approches sont caractérisées selon les critères suivants [Hachani, 2006] :

- **couverture :** tous les types de préoccupations sont ils considérés avec la solution proposée ?
- **expressivité :** le programme résultant est-il lisible et facile à comprendre ? La solution proposée pour la représentation des préoccupations est-elle explicite et modulaire ?
- **intégrabilité :** la solution proposée permet-elle de générer un programme lisible, facile a comprendre, à faire évoluer et à réutiliser ?
- **adaptabilité :** la solution proposée permet-elle une mise à jour facile (ajout, suppression et modification) de certaines préoccupations, sans autant modifier l'application de base ?

1.5 La séparation des préoccupations et le paradigme Aspect

Les limitations dont souffre la programmation orientée objet ont poussé les chercheurs en génie logiciel à préconiser une réelle *séparation des préoccupations* qui assure, entre autres, une meilleure modularité et réutilisabilité du code source. Ces travaux de recherche ont produit la programmation orientée aspect (POA), qui propose d'encapsuler dans des aspects les préoccupations incompatibles avec la logique métier des objets. Le code nécessaire à ces préoccupations est représenté par des aspects qui injectent ensuite le code nécessaire à ces préoccupations, en utilisant leurs propres mécanismes (points de coupure, greffons et introductions).

Le paradigme aspect est un paradigme qui trouve ses racines en 1996. Plus exactement la programmation orientée aspect (POA) est née suite aux travaux de Gregor Kiczales et de son équipe au centre de recherche de Xerox. La programmation orientée aspect est une technologie relativement jeune, car ses premiers outils ne sont apparus que fin 1998. L'idée principale de la programmation orientée aspect était de rendre le code source du programme plus lisible et beaucoup plus réutilisable, en implémentant de manière séparée les préoccupations transversales. A ce titre, il est important de donner une définition de la programmation orientée aspect.

> *Aspect oriented software development is a new technology for separation of concerns (SOC) in software development. The technique of AOSD make it possible to modularize aspects of a system* [Filman et al., 2004].

Dans la programmation orientée objet, la plupart des logiciels actuels répondent à diverses exigences. Ces exigences sont réparties en deux grandes catégories : les exigences fonctionnelles et les exigences non fonctionnelles ou techniques. Les exigences fonctionnelles décrivent le comportement du système et définissent les fonctions ou services que le système doit remplir, quant aux, exigences non fonctionnelles, elles expriment souvent les qualités ou contraintes imposées sur la manière de satisfaire les exigences fonctionnelles.

Le paradigme objet et ses langages dérivés posent fréquemment problème avec l'implémentation des exigences non fonctionnelles, du fait qu'il est difficile d'en limiter la portée dans un domaine bien circonscrit. On constate donc la présence de problématiques relatives à l'implémentation des exigences non fonctionnelles qui se trouvent dispersées dans les différents modules fonctionnels du système. C'est ce qu'on appelle des *préoccupations transverses (crosscuting concerns)*. Ces méthodes ont cependant fait l'objet de sérieuses critiques. Leur incapacité à implémenter correctement les préoccupations transversales, telles la journalisation ou la sécurité entraine, entre autres, une mauvaise modularité des programmes orientés objet, et les rend difficilement réutilisables. C'est dans le but de palier ces problèmes et de réaliser une réelle *séparation des préoccupations* [Parnas, 1972] qu'est né le nouveau paradigme de la programmation orientée aspect [Kiczales and Hilsdale, 2001].

La section précédente nous a permis de montrer que les modèles classiques de programmation présentent, le plus souvent, certaines limites et problèmes conséquents vis-à-vis de la séparation et de la représentation explicite et modulaire des préoccupations transversales. De la même façon ils n'offrent pas, ou peu, de solutions efficaces pour la composition et l'adaptation de telles préoccupations. Ces observations sont aujourd'hui au cœur de plusieurs travaux de recherche, dont le but principal est d'offrir des modèles permettant une meilleure séparation et composition de tout type de préoccupations. De nouveaux modèles et langages de programmation associés sont ainsi actuellement proposés. Ces modèles sont connus sous la désignation plus générale d'approche Aspect. Ils prônent une décomposition des programmes non seulement en unités modulaires représentant les préoccupations fonctionnelles de base, mais aussi en unités modulaires dédiées à la représentation des préoccupations transversales. Ils offrent en plus des solutions adéquates de composition de préoccupations (on parle aussi de tissage), afin de construire des systèmes efficaces.

Les modèles et langages de programmation, introduits dans le cadre de l'approche Aspect, sont définis le plus souvent comme étant des déclinaisons (ou des extensions) des modèles et langages de programmation déjà existants (procéduraux et fonctionnels, Objets, Composants, etc.). Les principaux modèles de programmation introduits dans le cadre de l'approche Aspect font l'objet de la section suivante.

1.5.1 La programmation adaptative

La programmation adaptative (Adaptative Programming ou AP) [Lieberherr et al., 1994], [Lopes and Lieberherr, 1994], [Palsberg et al., 1995], [Lieberherr et al., 2001], dite aussi programmation orientée patron, avec son implémentation DemeterJ [Palsberg et al., 1997] tente de résoudre les problèmes de dépendances entre comportements et structures d'objets d'une application, afin d'assurer une meilleure séparation des préoccupations.

En effet, toute collaboration dans un programme à base d'objets est assurée par un ensemble de méthodes, qui interagissent entre elles, en mettant en jeu plusieurs propriétés structurelles. Cette collaboration augmente considérablement les dépendances entre objets utilisés dans cette collaboration. Ces interactions rendent de telles collaborations très complexes et difficiles à comprendre. En effet, toute modification de ces collaborations provoque la modification d'un nombre important de méthodes, et affecte donc plusieurs classes de l'application. Cependant, la maintenance, l'évolution et la réutilisation des programmes deviennent difficile à effectuer. Par conséquent, toute modification dans la structure des objets de l'application nécessite une refonte de la collaboration.

Afin d'apporter des solutions à ces problèmes, la programmation adaptative modélise les préoccupations transverses dans des patrons qui sont classés en différentes catégories telles que les patrons de propagation d'opérations sur les données et les patrons de synchronisation d'accès concurrents. Cette structure permet de garder séparés et encapsulés, dans des unités modulaires distinctes, les comportements des différentes collaborations entres objets.

1.5.2 La programmation par rôles ou par points de vue

Dans la programmation par rôles ou par points de vue [Kristensen and Olsson, 1996], [Bardou and Dony, 1996], [McDirmid and Hsieh, 2003], les programmes sont structurés en groupes d'objets. Ces groupes représentent les différents points de vues du système à développer. Ce qui différencie cette approche de l'approche objet, est qu'une entité du monde réel, dans cette approche, est représentée par un ou plusieurs objets. Selon le point de vue considéré, une entité peut jouer plusieurs rôles. Chaque objet modélise un rôle particulier que peut jouer une entité. L'ensemble des préoccupations de l'application représentent les différentes vues subjectives du système. Chaque préoccupation est représentée par les différents rôles et leurs interactions. Il est à signaler que la distinction explicite entre préoccupations de base et préoccupations transversales n'est pas faite dans ce paradigme.

La programmation par rôles ou par points de vue présente aussi certaines limites liées à la séparation des préoccupations à cause des problèmes de partage de données et de recouvrement de diverses définitions entre préoccupations distinctes. En plus, il est à

signaler, la difficulté de mise en œuvre de l'intégration de propriétés d'objets représentant une même entité du monde réel mais appartenant à des groupes d'objets différents.

1.5.3 La programmation par sujet

La nouvelle technique de la séparation des préoccupations introduite par la programmation par sujets (Subject Oriented Programming ou SOP) [Harrison and Ossher, 1993], consiste à structurer les programmes en sujets représentant les différents développeurs, utilisateurs du système, et les contextes d'utilisation.

Dans cette approche, chaque sujet représente une préoccupation particulière. Un sujet est défini comme étant une collection d'états et de comportements d'un groupe d'objets/classes ou fragments de classes liés entre eux grâce à l'héritage ou aux autres relations. Ces objets reflètent une perception du monde réel. Dans ce contexte, la composition (connue aussi sous le nom d'intégration) de l'ensemble des sujets considérés produit le système final.

Pour effectuer l'intégration de l'ensemble des sujets trois types de règles de composition sont utilisés [Ossher et al., 1995], [Ossher et al., 1996]. Ces règles sont : les règles de mise en correspondance, les règles de combinaison, et les règles de combinaison et de mise en correspondance en même temps. Chaque règle de composition concerne plusieurs sujets destinés à être intégrés. Enfin, la définition des règles de composition est encapsulée dans un ou plusieurs modules de composition.

Ce paradigme offre un certain nombre d'avantages dans la conception et l'implémentation de systèmes complexes [Hachani, 2006] :

- il permet le développement d'objets décentralisés [Harrison et al., 1994]. Cependant les objets partagés par les développeurs peuvent avoir des définitions relatives aux vues subjectives des développeurs.
- un unique sujet cohérent définit une seule préoccupation.
- les fonctionnalités d'un système peuvent être étendues par le biais d'un nouveau sujet encapsulant l'évolution attendue (modifications ou extensions), à condition de préciser les règles de composition.

1.5.4 Les filtres de composition

L'approche par filtres de composition (*Composition Filters ou CF*) [Aksit and Tripathi, 1988], [Tripathi et al., 1989], [Aksit et al., 1994], [Aksit, 1996] considère l'objet comme une entité qui réalise un certain traitement. Elle est motivée par la difficulté d'adresser de manière indépendante et séparée la coordination de messages complexes échangés entre objets en collaboration. En effet, ces échanges de messages visent à réaliser une ou plusieurs préoccupations particulières d'un système. En se basant sur ce

constat, Aksit et al. [Aksit, 1996] proposent d'étendre le paradigme Objet en ajoutant le concept de filtre de messages, qui intercepte l'envoi et la réception de tout message envoyé ou reçu par un objet. Les filtres interceptent les messages et les manipulent, en modifiant leur portée et les comportements prévus. Alors que la portée fait référence à la délégation du message à d'autres objets, par ailleurs, le changement des comportements prévus est effectué par la substitution des sélecteurs des messages.

L'approche par filtres de composition considère deux types de filtres : les filtres d'entrée et les filtres de sortie. Les filtres d'entrée interceptent les messages reçus, quant aux messages envoyés ils sont interceptés par les filtres de sortie. Un message envoyé doit passer par tous les filtres d'entrée avant qu'il soit reçu par le destinataire. De la même manière, un message ne peut être envoyé qu'après avoir passé tous les filtres de sortie appliqués à l'objet émettant ce message. Chaque filtre représente une préoccupation particulière. L'approche par filtres de composition demeure une approche spécifique avec des concepts et un style particulier [Aksit, 1996].

1.6 Les concepts de base de la programmation orientée Aspect

Chaque nouveau paradigme de programmation propose un ensemble de nouveaux concepts. La programmation procédurale apporte les notions de module, fonction et de procédure. La programmation objet apporte les notions de classe et d'héritage, d'objet, de méthode. A son tour, la programmation orientée aspect (POA) apporte aussi des nouveaux concepts à savoir les concepts d'aspect, de point de jonction, de coupe et de greffon, qui seront présentés, de manière indépendante de l'implémentation, dans les sections suivantes.

1.6.1 La notion d'Aspect

L'apport principal de la programmation orientée aspect est le concept d'aspect lui même. Il est proposé pour capturer les différentes préoccupations transversales d'un système, telles la journalisation et la persistance. A cet effet, *Un Aspect est une entité logicielle qui capture une fonctionnalité transversale.* [Kiczales et al., 1997].

En POA, une application comporte des classes et des aspects. Un aspect se différencie d'une classe par le fait qu'il implémente une fonctionnalité transversale à l'application, qui se retrouve dispersée dans des classes en POO, et dans des modules, dans la programmation procédurale. La figure 1.1 [Smacchia and Vaucouleur, 2003] illustre cette

capacité. La partie gauche de cette figure représente le code des méthodes d'un programme orienté objet. La partie droite montre le même programme restructuré grâce à la POA. On peut voir que le code des méthodes a été nettoyé pour n'y laisser que la logique métier, et les autres préoccupations transverses sont encapsulées dans des aspects.

Le programmation par aspect permet, par le biais de ces mécanismes, d'intervenir au niveau d'un programme orienté objet dans un point quelconque de l'exécution (ie : les points de jonction), pour injecter le comportement nécessaire à la réalisation de la préoccupation transversale.

1.6.2 La notion de point de jonction

Un point de jonction (pointcut) se définit comme *un point dans l'exécution d'un programme autour duquel un ou plusieurs aspects peuvent être définis.* Pour désigner de manière concrète cette notion de point de jonction dans un code d'aspect, le concept de coupe est utilisé.

1.6.3 La notion de coupe

Selon [Renaud et al., 2004] : *une coupe désigne un ensemble de point de jonctions.* Une coupe est définie à l'intérieur d'un aspect. Dans le cas simple, une seule coupe suffit pour définir la structure transversale d'un aspect, dans les cas plus complexes, un aspect est associé à plusieurs coupes. Les notions de coupe et de point de jonction sont liées par leur définition. Pourtant, leur nature est très différentes. Une coupe est un choix de points de jonction défini dans un aspect, alors qu'un point de jonction est un emplacement dans l'exécution d'un programme. De ce fait un point de jonction peut appartenir à plusieurs coupes dans un même aspect ou dans des aspects différents [Renaud et al., 2004].

1.6.4 La notion de greffon

Le greffon (advice) *définit ce que l'aspect greffe dans l'application*, autrement dit, les instructions ajoutées par l'aspect. Un aspect définit un ou plusieurs greffons, chaque greffon réalise un comportement particulier de son aspect. Le code greffon joue en quelque sorte le même rôle qu'une méthode. A la différence des méthodes, les greffons sont associés à une coupe, et donc à des points de jonction, et ont un type qui détermine le moment de leur exécution (before, after, arround).

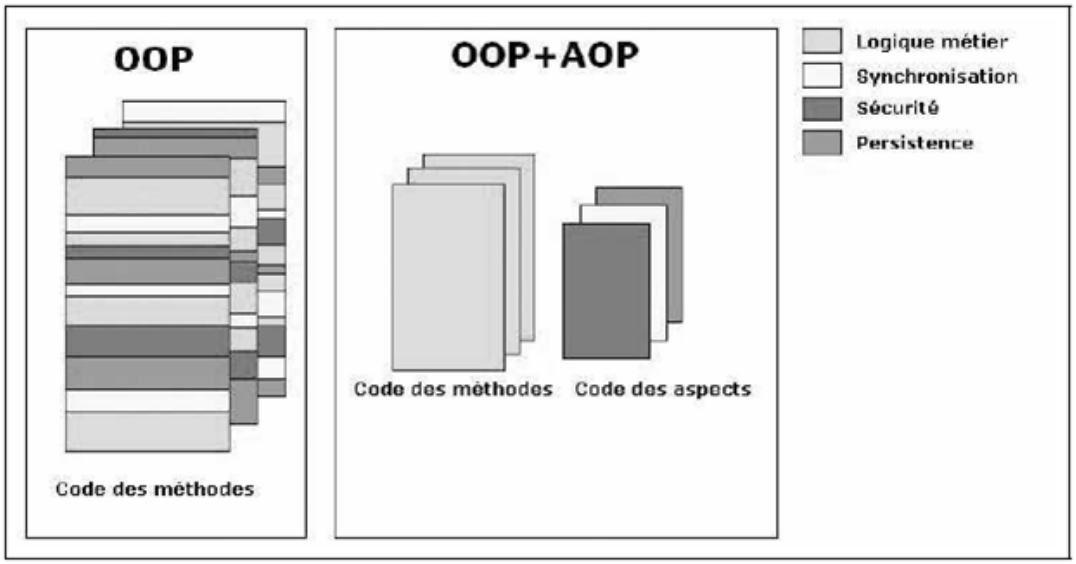


Figure 1.1 — L'encapsulation des préoccupations transversales [Smacchia and Vaucouleur, 2003]

1.7 Un langage de programmation orienté Aspect : AspectJ

La programmation par aspect s'applique aux langages orientés objet de façon générale. Il existe des implémentations spécifiques à des langages tel que C++ (AspectC++ [Spinczyk et al., 2002]) ou encore C#. AspectJ est l'une des implémentations du paradigme orienté Aspect appliquée au langage Java. Les sections suivantes présentent les principaux mécanismes et abstractions qui composent AspectJ [Renaud et al., 2004]. Les définitions des concepts d'AspectJ, présentés dans les sections suivantes, sont extraites du livre *"Aspect-Oriented Software Development"* [Filman et al., 2004] et celui de Pawlak et al. intitulé *"La programmation orientée aspect pour Java/J2EE"* [Renaud et al., 2004]

1.7.1 Point de jonction

Le point de jonction (joinpoint) est le concept qui permet de définir un point précis dans l'exécution d'un programme pour injecter une action. AspectJ permet de spécifier plusieurs types de points de jonction [Gradecki and Lesiecki, 2003] :

- appel et exécution de méthode.
- appel et exécution de constructeurs.

– accès en lecture ou en écriture à un attribut.
– exécution d'un gestionnaire d'exception.
– initialisation d'une classe ou d'un objet.

AspectJ permet de choisir les points de jonction, associés à un aspect, grâce au mécanisme de coupure.

1.7.2 Point de coupure

Le point de coupure, déclaré grâce au mot clé `Pointcut`, est le mécanisme qui permet de déclarer les points de jonction utilisés dans un aspect. L'extrait suivant montre la déclaration d'un point de coupure, appelé `Pointmoved`. Ce point de coupure représente un appel à une méthode préfixée par `set` de la classe `Point`. De plus cette méthode est de type public et retourne la valeur `void`. Il est mentionné que le point de coupure fait référence à un appel de méthode grâce au prédicat `call`.

```
Pointcut pointmoved () : call(public void Point.set*(int))
```

1.7.3 Greffon

Un Greffon (Advice) permet d'encapsuler le code à exécuter et de l'associer à un point de coupure. Il est de trois types :

– `After`
  le greffon est exécuté après le point de coupure associé.
– `Before`
  le greffon est exécuté avant le point de coupure associé.
– `Around`
  le greffon est exécuté à la place du point de coupure associé.

1.7.4 Mécanisme d'introduction

Le mécanisme d'introduction autorise le développeur à déclarer des membres (méthodes ou attributs) dans des classes déjà existantes ou de modifier la hiérarchie des classes et des interfaces.

1.7.5 Aspect

Les aspects sont une unité modulaire permettant à AspectJ d'implémenter des préoccupations transversales (crosscutting concerns). Les Aspects sont déclarés par le mot clé `aspect` et peuvent contenir des méthodes et des attributs (au même titre qu'une classe), point de coupure, greffon, et des mécanismes d'introduction (inter-type declarations).

1.7.6 Tissage

Le tissage (Weaving) est l'étape concrète qui permet d'injecter le comportement décrit par les Aspects dans un programme orienté objet. Le tissage peut se faire de manière statique comme le fait AspectJ grâce à son compilateur `ajc` ou de manière dynamique à l'exécution du programme.

1.7.7 Exemple de programme en AspectJ

Pour illustrer les différents concepts définis dans AspectJ, nous prenons, comme exemple, un extrait d'un programme en AsjectJ décrit dans [Renaud et al., 2004], ce programme gère les commandes d'articles d'un client. Il permet à un client d'ajouter des articles à une commande et de calculer le montant. Pour cela un catalogue d'articles est utilisé. Ce catalogue mentionne les prix des différents articles qui peuvent être commandés. L'application est constituée de trois classes : `Customer, Order et Catalog`. Notons, que la classe `Customer` est le point d'entrée de l'application. Son code est le suivant [Renaud et al., 2004].

```
package aop.aspectJ;
public class Customer{
        public void run(){
        Order myOrder= new Order();
        myOrder.addItem("CD", 2);
        myOrder.addItem("DVD", 1);
        double montant = myOrder.computeAmount();
        System.out.println("Montant de la commande: "+montant+"euros");
}
public static void main(String[] args)
        new Customer().run();
}
}
```

Le rôle de la méthode `main()` est de créer un objet de la classe `Customer`, puis elle appelle la méthode `run` . Cette méthode crée, d'abord, une commande (objet `myOrder`), appelle deux fois la méthode `addItem` avec comme paramètres une référence et une quantité, et puis calcule le montant de la commande afin qu'il soit affiché.

L'affichage de l'ensemble des articles commandés est attribué à un Aspect qui affiche un message avant et après les appels de la méthode `addItem` de la classe `Order` comme le montre l'extrait du programme suivant [Renaud et al., 2004].

```
package aop.aspectJ;
public aspect traceAspect{
pointcut: ToBeTraced():
        call (public void Order.addItem(string,int));

void around(): ToBeTraced {
        System.out.println("Avant Appel addItem");
        proceed();
        System.out.println("Après Appel addItem");} }
```

Cet aspect contient un pointcut `ToBeTaced` composé d'un seul point de jonction `(call (public void Order.addItem(string,int))` qui capture l'appel de la méthode `addItem` de la classe `Order`. Le greffon (`(void around(): ToBeTraced)` utilisé est de type `around` attaché au point de coupure `ToBeTaced` [Renaud et al., 2004].

1.8 Conclusion

Ce chapitre nous a permis, dans un premier temps, de présenter le concept de la séparation des préoccupations, ainsi que le concept des préoccupations transversales. Il passe en revue les approches de programmation appliquant ce principe dans le développement de logiciels. Ces approches sont classées en deux grandes catégories : les approches classiques et les approches Aspect. Les approches classiques (Procédures, modules et objets, programmation réflexive, générique et les approches à base de composants), qui présentent leur lot d'inconvénients face au problème des préoccupations transversales. Nous avons montré comment est pris en charge le problème des préoccupations transversales avec l'approche Aspect, la programmation adaptative, par rôle ou par point de vue, la programmation par sujet ainsi que les filtres de composition.

Dans un second temps, ce chapitre présente le paradigme Aspect et ses concepts de base, indépendamment de tout langage orienté aspect. Puis une partie est réservée pour présenter un langage de programmation orienté aspect celui d'AspectJ, considéré comme le plus connu dans cette catégorie de langages de programmation. Un exemple est donné en fin de chapitre, pour illustrer les concepts de la POA sous AspectJ.

Dans le chapitre suivant, nous étudions les approches d'ingénierie des exigences classiques, et celles orientées Aspect.

Deuxième partie

L'Ingénierie des Exigences orientées Aspect

2 L'ingénierie des exigences orientée Aspect

Users don't know what they want until you show it to them (Kent Beck).

2.1 Introduction

L'ingénierie des exigences (IE) s'applique au processus d'ingénierie des systèmes, elle s'applique également au processus logiciel et au processus permettant de créer les architectures matérielles. Dans notre travail, nous nous intéressons à l'IE qui est la première étape fondamentale dans n'importe quel cycle de développement logiciel. Cette étape est primordiale, et doit être menée avec beaucoup d'attention.

Les motivations pour l'IE sont multiples, en 1976, Bell & Tayer [Bell and Thayer, 1976] ont constaté que l'inadéquation des fonctionnalités du système aux besoins des usagers, l'incohérence, l'incomplétude et l'ambiguïté des documents des exigences sont en partie à l'origine de la mauvaise qualité du logiciel final.

Plusieurs enquêtes ont confirmé l'ampleur des dégâts causés par l'insuffisance de qualité des documents des exigences. Une enquête menée auprès de 800 projets conduits dans 350 compagnies américaines par le Standish Group (2003) et présentée dans deux rapports, intitulés *Chaos* et *Unfinished Voyages*, a révélé que 31% des projets sont annulés avant même d'être terminés. En 1995, cela a couté 81 milliards de dollars aux compagnies américaines. Ce même rapport montre que 50% d'entre eux n'avaient que partiellement réussi, dans le sens où ils avaient nécessité des budgets et des délais très fortement majorés. La mauvaise qualité des documents des exigences constitue 47% des causes d'échecs cités. Le rapport de l'année 2009 du Standish Group [Standish-Group, 2009] montre que 24% des projets sont abandonnés ou jamais utilisés, 44% sont partiellement réussis et que simplement 32% des projets sont réellement utilisés.

Afin de minimiser le taux d'échec de l'utilisation des systèmes informatiques, l'étape

de l'IE doit être clairement identifiée dans le processus de développement, et l'application des méthodes et des technologies d'IE s'avère nécessaire.

Après avoir présenté, dans le premier chapitre, le paradigme Aspect, qui représente conjointement avec le concept d'IE, le cadre général de note travail, ce chapitre présente l'IE de façon générale et l'Ingénierie des Exigences orientée aspect en particulier. Il est organisé comme suit. Dans la section 2.2 nous présentons quelques définitions de l'IE. Le processus d'IE est ensuite présenté dans la section 2.3. La section 2.4 détaille quelques approches traditionnelles d'IE. Ensuite, nous abordons les approches d'IE orientées Aspect connues sous le vocable anglais Aspect oriented requirements engineering (AORE) dans la section 2.5. La dernière section établit une conclusion.

2.2 Ingénierie des exigences : définitions

Avant de définir le concept d'IE, nous commençons par la définition des concepts de ***partie prenante***, ***besoin***, et ***exigence***.

> *Une* ***partie prenante*** *peut être une entreprise, une organisation ou un individu ayant un intérêt ou une partie dans le résultat de l'ingénierie d'un système. Selon l'AFIS (Association Française d'Ingénierie Système), une partie prenante constitue une partie intéressée par l'utilisation et l'exploitation du système (voire par ses impacts sur son environnement), il peut être aussi un agent participant à sa conception, sa production, son déploiement, sa commercialisation, son maintien en condition opérationnelle et son retrait de service. [GTIE, 2000].*
>
> *Un* ***Besoin*** *est la perception que l'utilisateur a du système [Didier, 2002]. Il est souvent exprimé sous forme de problèmes que rencontrent les utilisateurs auxquels le système est destiné.*
>
> De fait, il n'existe pas une définition unique de *l'****Exigence***, mais plusieurs telle que celle donnée par Ross [Ross and Schoman, 1977] en 1977, qui définit une ***Exigence*** *comme une évaluation précise du besoin auquel le système devra répondre. Les exigences doivent exprimer pourquoi le système est requis et préciseront quelles sont les fonctionnalités du système qui serviront et satisferont ce contexte. Elles doivent également préciser comment le système est construit.*
>
> Davis considère une ***exigence*** *comme étant une caractéristique visible, vue de l'extrieur de système [Davis, 2005].*
>
> Selon l'IEEE, une ***exigence*** *est une condition ou une capacité qui doit être satisfaite par un système ou composant d'un système pour satisfaire un*

contrat, une norme, une spécification, ou autres documents formellement imposés (IEEE 1990). Comparativement au besoin, qui est lié à l'utilisateur, l'exigence est la vision que le concepteur ou le développeur a du système [Didier, 2002].

Selon Gyger, l'IE est concernée par l'identification de buts assignés au système envisagé et l'opérationnalisation de ces buts en contraintes et exigences imposées au système et aux agents qui en assureront le fonctionnement . Le Groupe de Travail sur l'Ingénierie des Exigences (GTIE) et l'Association Française d'Ingénierie système(AFIS) [AFIS, 2007], définissent l'IE comme étant l'ensemble des activités destinées à découvrir, analyser, valider et faire évoluer un ensemble d'exigences relatives à un système. Elle permet de montrer la satisfaction des exigences et des engagements durant tout le cycle de vie d'un système [GTIE, 2000].

L'IE est donc une activité permettant de prendre en compte de manière systématique et formalisée les exigences des parties prenantes d'un système. Cette prise en compte est indispensable pour la réussite du développement et de l'utilisation du système.

2.3 Le processus d'ingénierie des exigences

L'IE est une étape essentielle, elle est à l'origine de toute activité de développement, elle est nécessaire pour surmonter les problèmes de communication, elle est composée généralement de quatre phases essentielles qui sont : l'élicitation ou la découverte des exigences, la négociation, la spécification et enfin la validation des exigences.

Certains auteurs proposent des découpages différents du processus d'IE : selon Nuseibeh & Easterbrook [Nuseibeh and Easterbrook, 2000], les phases d'élicitation, de modélisation, d'analyse, de spécification, de validation et de gestion des exigences constituent les phases du processus d'IE ; Kotonya & Sommerville [Kotonya and Sommerville, 1998] ont mis en évidence toutes les phases précitées sauf celle de la modélisation.

Ces différences ne s'expliquent pas forcément par une omission de phases, mais peuvent être considérées comme différentes façons de voir le processus, par exemple Kotonya & Sommerville [Kotonya and Sommerville, 1998] incluent la modélisation dans la phase d'analyse.

Ces différentes phases se définissent comme suit :

- ***L'élicitation*** des exigences consiste en la collecte, la capture, la découverte et le développement des exigences à partir d'une variété de sources y compris les parties prenantes humaines.
- ***La modélisation*** est une abstraction du problème par des représentations parfois graphiques tels que les modèles de cas d'utilisation d'UML ou les modèles

entité-association. C'est la construction d'une description et d'une représentation abstraite d'un système pour les besoins de l'interprétation [Nuseibeh and Easterbrook, 2000].

- ***L'analyse*** consiste à vérifier certains concepts tels que l'exactitude, la complétude, la clarté et la consistance des exigences. Pour cela, l'analyse peut se baser sur les résultats issus de la phase de modélisation dans le but de clarifier les exigences, de supprimer les incohérences, et d'assurer la complétude et la non redondance.
- ***La spécification*** vise à documenter les exigences, elle est importante pour les développeurs qui doivent concevoir et construire le système. Elle consiste à organiser les exigences par catégorie, et ceci dans le but d'avoir une vision commune du système. La catégorisation des exigences peut se faire en bénéficiant des apports des méthodes orientées aspect [Awais et al., 2003].
- La ***vérification et la validation*** des exigences consiste à examiner le document des exigences pour assurer qu'il est cohérent, complet et non ambigu, et que les parties prenantes sont satisfaites par la spécification finale des exigences.
- ***La gestion des exigences*** couvre tout le processus d'IE. Elle consiste à assurer la traçabilité pour contrôler les différentes évolutions ou le changement des exigences [Wiegers, 2003].

Les principaux concepts des approches de l'IE sont : but, point de vue, scénario, agent et aspect. D'autres approches exploitent à la fois les concepts de but et de scénario pour définir une autre catégorie d'approche d'IE dite approche hybride. Nous classons les approches d'IE en deux grandes catégories qui sont : les approches d'IE non orientées Aspect et les approches orientées Aspect. Les sections suivantes présentent en détails chacune de ces approches.

2.4 Les approches d'ingénierie des exigences non orientées Aspect

Cette section présente les principales approches non orientées Aspect, celles qui sont orientées par les buts, les scénarios, les points de vue et les approches hybrides.

2.4.1 Les approches orientées but

La notion de but est de plus en plus utilisée dans les méthodes et les techniques actuelles de l'IE. Les buts ont été introduits pour plusieurs raisons, notamment pour atteindre différents objectifs dans les activités de l'IE. Les buts contribuent substantiellement à l'acquisition des exigences. L'identification des buts conduit naturellement à poser répétitivement les questions "Pourquoi ?", "Comment ?" et "Comment autrement ?".

Selon A. Van Lamsweerde [Lamsweerde, 2000], un but est un objectif que le système futur doit réaliser : les buts sont classés en deux niveaux.

- Les buts de haut niveau englobent les buts stratégiques, globaux ou les buts couvrant l'organisation.
- Les buts de bas niveau décrivent les buts opérationnels, locaux ou spécifiques à la conception.

Les buts représentent les propriétés attendues et peuvent couvrir aussi bien les exigences fonctionnelles à fournir par le nouveau système que les exigences non fonctionnelles liées à sa qualité de service, comme la performance, la sécurité, etc.

Les buts peuvent être utilisés pour fournir une analyse sur les exigences, en évaluant la complétude des exigences et leur pertinence aussi bien que leur identification. Par exemple, une exigence est justifiable et pertinente si elle mène à la satisfaction d'un but. L'ensemble des exigences est complet s'il satisfait tous les buts.

Dans les sections suivantes, nous présentons avec plus de détails l'approche NFR framework [Kotonya and Sommerville, 1998] comme un exemple d'approche d'IE orientée but et nous résumons deux autres approches KAOS [Lamsweerde et al., 1991] et i* [Yu, 1994].

2.4.1.1 L'approche NFR framework

L'approche NFR framework [Kotonya and Sommerville, 1998] est qualifiée d'approche d'analyse des exigences non fonctionnelles (NFR), qui considère qu'une exigence est une description d'un service du système ou une contrainte qui doit être prise en compte par le système afin de réaliser un but. Ainsi, une exigence spécifie comment un but doit être accompli par un système donné.

Dans [Loucopoulos and Karakostas, 1995], l'approche NFR framework est développée pour représenter et analyser les buts non fonctionnels. Cette approche présente le concept essentiel de `softgoal`.

Les `softgoals` représentent les exigences non fonctionnelles. Ils sont dépendants les uns des autres. Ces rapports de dépendance sont utilisés pour déduire comment un `softgoal` est satisfait. Les `softgoals` peuvent être raffinés en les décomposant en sous `softgoals` en utilisant des opérateurs (`AND/OR`) avec une sémantique évidente. Aussi, les interdépendances des `softgoals` peuvent être représentées avec des signes de contribution positifs(+) ou négatifs (-). Dans l'approche NFR framework on trouve les cinq composants : `softgoal`, `interdépendances`, `procédures d'évaluation`, `méthodes et corrélations` [Loucopoulos and Karakostas, 1995].

Un `softgoal` n'a pas de définition exacte et/ou de critère permettant de savoir s'il a été satisfait ou non. Les `softgoals` sont identifiés et utilisés pour représenter les

exigences. Une fois identifiés les `softgoals` sont reliés entre eux par des liens d'interdépendance (père-fils) sous la forme d'un graphe d'interdépendance avec la précision du type d'influence (positif ou négatif). Dans le cas d'une contribution positive, la satisfaction du `softgoal` père mène à la satisfaction du fils, par contre dans une contribution négative la satisfaction du fils mène à l'insatisfaction du père.

Pour décomposer un `softgoal`, cette approche utilise le `catalogue` décrivant le processus de raffinement par décomposition. La figure 2.1 illustre la décomposition du `softgoal` de sécurité d'un compte bancaire. Les `softgoal` peuvent être opérationnalisés et augmentés pour produire le graphe d'interdépendance des `softgoals` (SIG). A titre d'exemple le `softgoal` *AccessAuthorisation* est décomposé en trois sous softgoals qui sont (*Authentification*, *AccessRuleValidation* et *Identification*). Tous ces sous softgoals doivent être satisfaits ensemble en utilisant l'opérateur `AND`. Le sous softgoal *Authentification* est décomposé à son tour en trois sous softgoals (*Biometrics*, *CadrKey* ou *Password* liés par l'opérateur `OR`, comme le montre la figure 2.2.

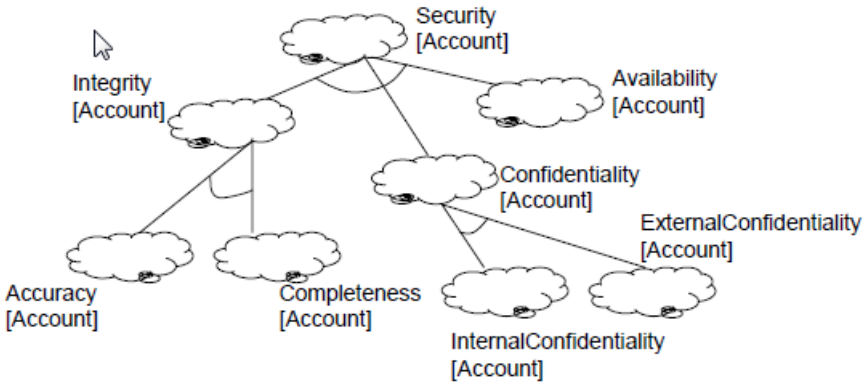


Figure 2.1 — Le catalogue de décomposition d'un softgoal en sous softgoals

Dans l'approche NFR framework, les `softgoals` identifiés doivent être catalogués et rangés sous forme de types de hiérarchies de relation `IsA` qui raffinent le `softgoal` initial. Ces `catalogues` sont destinés à une future réalisation et évitent l'oubli des exigences importantes de l'application. Ces `catalogues` sont : le `catalogue NFR framework`, le `catalogue de méthode` et le `catalogue de règles de corrélation`.

- `Le catalogue des softgoals` : il inclut des informations sur les types particuliers de `softgoals` (e.g. performance, sécurité, etc).
- `Le catalogue des méthodes` : les connaissances qui sont utilisées pour raffiner un `softgoal` sont enregistrées dans ce catalogue ainsi que l'opérationnalisation. Ce catalogue peut avoir une méthode générique qui déclare qu'un `softgoal` appliqué à un élément peut être décomposé en `softgoals` pour tous les composants de cet

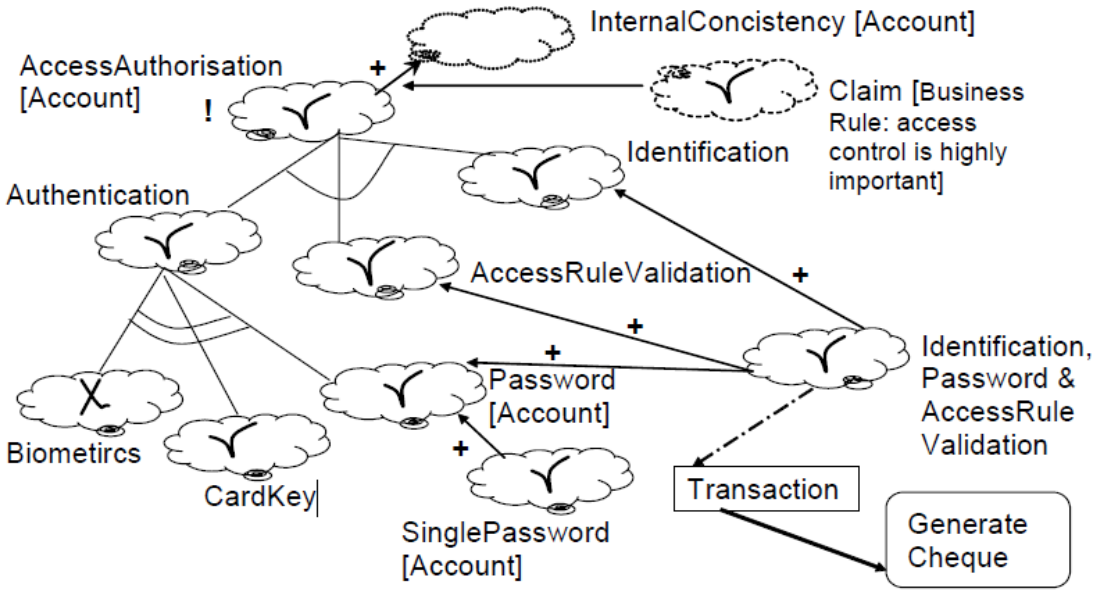


Figure 2.2 — Le graphe partiel d'indépendance du softgoal *internalConsistancy* d'un compte bancaire [Jethro et al., 2005]

élément.

– `Le catalogue des règles de corrélation` : il contient des données qui aident à la détection des interdépendances implicites entre les `softgoals`. Ce catalogue peut indiquer que l'indexation contribue positivement au temps de réponse.
La figure 2.1 présente un exemple du `catalogue des softgoals` qui porte sur le `softgoal` *Security*.

L'approche NFR framework met en évidence l'importance des exigences non fonctionnelles, qui sont transversales par nature. L'approche ne clarifie pas avec exactitude comment les exigences non fonctionnelles sont identifies (obtenues), suggérant seulement qu'elles doivent être obtenues en recueillant les connaissances du domaine pour lequel le système est construit. Le graphe des dépendances (SIG) encapsule le traitement de chaque exigence non fonctionnelle.

2.4.1.2 Autres méthodes d'ingénierie des exigences orientées but

Plusieurs autres méthodes d'IE orientées but existent dans la littérature, telles que : la méthode KAOS (Knowledge Acquisition for autOmatted System) [Lamsweerde et al., 1991] et i* (i star) [Yu, 1994], etc.

KAOS est constituée d'un langage de modélisation, une méthode et un environnement

logiciel. Elle prend en considération presque toutes les activités essentielles de l'IE, que sont : l'élicitation, l'analyse, la documentation, la vérification et la validation de buts. Elle se concentre principalement sur l'élicitation de buts à différents niveaux d'abstraction [Jiang, 2005], et [Jureta, 2005].

Selon Ivan Jureta [Jureta, 2005], KAOS est conçue pour établir des descriptions précises des problèmes en utilisant des concepts prédéfinis et applicables, en analysant ces problèmes par le moyen d'une approche systémique pour découvrir et structurer les exigences. L'un des objectifs les plus importants de cette approche est de clarifier les responsabilités pour toutes les parties prenantes dans le projet et leur permettre de communiquer efficacement au sujet des exigences. Selon Jiang [Jiang, 2005], KAOS est destinée aux petits projets, d'une complexité moyenne, avec une équipe de développement de taille moyenne.

La méthode i* [Yu, 1994] fournit une description de l'organisation du travail en termes de relations de dépendances entre acteurs. Elle reconnait le fait que les acteurs ont une liberté d'action dans les contraintes sociales (inter acteurs) qui sont appelées dépendances stratégiques (strategic dependency - SD -). i* est fondée sur les sept concepts suivants : Acteur (Actor), But (Goal), Croyance (Belive), Plan (Plan), Ressource (Resource), Capacité (Capacity) et Dépendance (Dependency).

- Acteur : ce concept modélise une entité ayant des buts stratégiques et des intentions. Il représente un agent physique (par exemple une personne, un animal, une voiture) ou un agent logiciel. Il peut également être un rôle ou une position. Un rôle est une caractérisation abstraite du comportement d'un acteur dans un certain contexte, alors qu'une position représente un ensemble de rôles, typiquement joués par un seul acteur.
- But : un but représente un intérêt stratégique d'un acteur. i* distingue les buts fonctionnels (`hard-goals` ou `goals`) des buts non-fonctionnels (`soft-goals`). Les `soft-goals` n'ont pas de définition ou critères de satisfaction. Ils sont utiles pour modéliser les qualités du futur système, telles que la sécurité, la performance et la maintenabilité.
- Croyance : les croyances sont utilisées pour représenter les connaissances de chaque acteur sur l'environnement.
- Plan : un plan représente une manière ou un ensemble d'actions réalisées pour satisfaire un but.
- Ressource : une ressource représente une entité physique ou informationnelle recherchée par un acteur et fournie par un autre acteur.
- Capacité : la capacité représente l'aptitude d'un acteur à définir, choisir et exécuter un plan pour réaliser un but, dans un environnement donné.
- Dépendance : une dépendance entre deux acteurs indique qu'un acteur dépend d'un autre acteur pour atteindre un but, exécuter un plan ou délivrer une ressource.

En dépendant d'un autre acteur, un acteur est capable de réaliser des buts qu'il ne pourrait pas ou pas facilement réaliser, en ne comptant que sur lui-même. Le dépendant devient vulnérable, si l'acteur dépend d'un autre acteur qui n'arrive pas à fournir l'objet de la dépendance. Dans ce cas-là, le premier acteur est affecté dans sa capacité d'atteindre ses buts.

La méthode i* classe les exigences en deux grandes catégories : les exigences préliminaires et les exigences finales. Durant l'analyse des exigences préliminaires, l'analyste modélise et analyse les intentions des acteurs. Dans i*, les intentions sont modélisées en buts qui, à travers une analyse orientée but, conduisent aux exigences fonctionnelles et aux exigences non fonctionnelles du futur système. L'élicitation des exigences dans i* est un processus composé de deux étapes :
- la première étape consiste à éliciter des exigences préliminaires en utilisant un diagramme de dépendance stratégique (SD).
- la deuxième étape a pour objectif l'élicitation des exigences finales en élaborant un diagramme de raisonnement (strategic rationale (SR)).

2.4.2 Les approches à base de scénarios

Les approches à base de scénarios font l'hypothèse qu'il est plus facile de décrire, le plus souvent dans un récit en langue naturelle, ce qui se passera dans le système futur que d'exprimer directement les objectifs de ce système. Les acteurs de l'organisation ont parfois des difficultés à énoncer à priori les objectifs attendus du système ; ceux-ci ne sont facilement explicités que lorsque les acteurs ont bien compris ce que doit faire le système. La difficulté à manipuler des notions abstraites comme les buts a été reconnue par des études en sciences cognitives [Kevin Benner and Zorman, 1993]. Un scénario est une séquence d'interactions entre le système envisagé et son environnement énoncée dans le contexte restreint d'un propos particulier. Il décrit une histoire, c'est un écrit concret de la façon dont un acteur imagine d'interagir avec le système. C'est pour cela que l'on s'est tourné vers les scénarios.

L'introduction des scénarios d'usages (use cases) [Jacobson et al., 1992] dans le monde de l'orienté objet confirme la tendance à l'utilisation de scénarios dans les méthodes de conception de systèmes. De nombreuses méthodologies sont maintenant proposées. Cependant, selon les méthodologies, le terme scénario porte des sémantiques bien différentes. On associe parfois les scénarios à des traces (d'évènements internes et/ou externes), des échanges de messages entre composantes, des séquences d'interactions entre un système et l'utilisateur, ou encore des ensembles plus ou moins génériques de telles séquences, pour ne mentionner que ces quelques définitions. Nombreuses sont aussi les notations utilisées, qu'elles soient basées sur une grammaire textuelle plus ou moins formelle, sur des automates, ou sur des diagrammes d'échanges de messages semblables aux

diagrammes de séquence des messages (MSC :Message Sequence Chart). Les approches diffèrent donc sur plusieurs points en fonction de la définition retenue.
Définition 1 : un scénario est une séquence de responsabilités (événements et activités), internes ou externes, qui sont reliées de façon causale dans le but d'offrir une certaine fonctionnalité.
Définition 2 : un scénario peut être défini comme l'ordre des actions ou des évènements intervenant dans un cas spécifique d'une certaine tâche générique qu'un système doit accomplir.

Les scénarios sont exprimés dans différentes notations : informelles, semi-formelles ou formelles. Les scénarios informels sont décrits à l'aide du langage naturel [Rolland et al., 1999],[Erickson, 1995],[Holbrook, 1990], ou de vidéos [Wood et al., 1994], [Haumer et al., 1998]. Ils sont utilisés lorsque les utilisateurs rejettent les notations formelles ou semi formelles [Weidenhaupt et al., 1998]. Les scénarios semi-formels utilisent des notations structurées comme des tableaux [Anton et al., 1994] ou des scripts [Rubin and Golberg, 1992]. Pour finir, les scénarios formels sont décrits à l'aide de langages basés sur des grammaires régulières [Glinz, 1995] ou des diagrammes d'états [Harel, 1987]. Ils peuvent être utilisés pour simuler le fonctionnement futur du système et juger des réactions des utilisateurs.

2.4.3 Les approches hybrides utilisant but et scénario

L'expérimentation des approches orientées but sur le terrain a montré des limites. Le concept de but reste flou, et s'il est admis que l'on démarre le développement d'un système avec un certain nombre d'objectifs en tête, la question se pose de l'origine des buts [Anton, 1996], de plus, les buts stratégiques qui initient le processus d'IE sont souvent idéalisés et irréalistes [Anton, 1996], [Anton et al., 1994] et peuvent conduire à des exigences inexactes. Les approches à base de scénario souffrent aussi de quelques inconvénients :

- la fragmentation des exigences saisies dans de multiples scénarios rend difficile l'assurance de complétude de la spécification des exigences [Lamsweerde, 2000].
- elle rend également difficile la recherche des différents aspects d'une même fonctionnalité du système à travers de multiples scénarios.
- le caractère "ad hoc" du processus d'identification des variations du scénario de base d'un cas d'utilisation.
- enfin, si la pratique montre que le processus d'IE à base de scénarios est conduit de manière descendante (top-down), elle met en évidence les difficultés rencontrées dans la maitrise des niveaux d'abstraction, un même scénario pouvant comporter des actions de différents niveaux d'abstraction et donc faire référence à des exigences de différents niveaux .

L'approche CREWS-l'Ecritoire (CREWS pour Cooperative Requirements Engineering With Scenario) est une approche d'IE qui tente de coupler les deux concepts de but et de scénario selon une relation bidirectionnelle. La méthode CREWS-l'Ecritoire, est une approche développée dans le cadre d'un projet ESPRIT (European Reactive Research Project), elle permet la découverte semi automatique des exigences à partir de scénarios textuels au moyen d'un couplage bidirectionnel entre but et scénario. L'Ecritoire est l'application logicielle de l'approche CREWS.

Afin de pallier les difficultés recensées ci-dessus, l'approche CREWS propose :

1. des directives méthodologiques pour apporter des conseils d'écriture des scénarios textuels (écrits en langue naturelle) et des outils logiciels pour vérifier leur correction.
2. des règles d'analyse des scénarios aidant à la découverte des variantes, cas d'exceptions et compléments d'un scénario donné et une formalisation du processus pour le systématiser tout en guidant son déroulement.

2.4.3.1 La notion de but dans la méthode CREWS

CREWS propose un formalisme de but dérivé d'une définition linguistique développée par [Prat, 1997]. Celle-ci est fondée sur l'étude d'un large échantillon de buts décrits dans la littérature, et sur le formalisme de la grammaire de cas [Fillmore, 1968], [Dik, 1989]. Selon ce formalisme, un but est composé d'un verbe suivi d'un ensemble de paramètres (une fonction sémantique est associe à chaque paramètre, une même fonction ne pouvant pas être associée à deux paramètres différents). Il existe quatre types de paramètres : **Cible**, **Direction**, **Voie** et **Bénéficiaire** comme le montre la figure 2.3.

Cible : la cible concerne les entités affectées par le but. Il y a deux types de cibles : *l'Objet* et le *Résultat*. L'Objet existe avant la réalisation du but et peut éventuellement être modifié ou supprimé par celui-ci, alors que le résultat est l'entité qui résulte de la réalisation du but désigné par le Verbe.

Direction : les deux types de direction sont appelés la *Source* et la *Destination* et identifient respectivement l'état initial et final de l'objet. La Source est le point de départ du but (source d'information ou lieu physique), et la *Direction* est son point d'arrivée.

Voie : une voie est spécialisée par les deux paramètres *manière* et *moyen*. La manière spécifie la façon d'atteindre le but et le Moyen est l'entité ou l'outil, par lequel le but est atteint.

Bénéficiaire : la personne ou le groupe en faveur de qui le but doit être atteint.

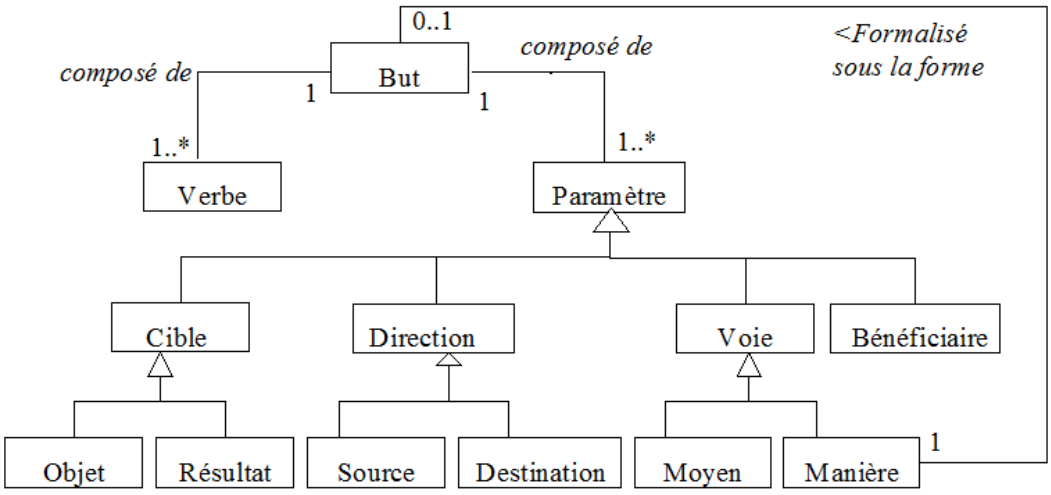


Figure 2.3 — La structure du but en notation UML dans l'approche CREWS

2.4.3.2 La notion de scénario dans la méthode CREWS

Dans CREWS, un scénario est défini comme *un comportement possible qui limite un ensemble d'interactions entre plusieurs agents* . Donc, un scénario est caractérisé par un état initial et un état final. Un état initial attaché au scénario définit une pré-condition pour que le scénario soit déclenché. Par exemple, le scénario *"Déposer les bouteilles dans une machine de recyclage à carte"* dans le cas normal ne peut pas être déclenché si l'état initial exprimé par les deux conditions : *"L'utilisateur a une carte"* et *"La machine de recyclage est prête"* ne sont pas vraies. Il y a deux types de scénarios : normaux et exceptionnels comme le montre la figure 2.4 . Le scénario normal permet d'atteindre le but associé, tandis que le scénario exceptionnel décrit le cas où le but n'est pas atteint. Le scénario associé au but *"Déposer les bouteilles dans une machine de recyclage à carte"* dans le cas exceptionnel quand "la carte n'est pas valide" est un exemple de scénario exceptionnel, car le but *"Déposer les bouteilles dans une machine de recyclage carte"* n'est pas atteint.

Le scénario peut être décrit en utilisant deux types d'actions : les actions atomiques et les flux d'actions. Une action atomique est une interaction entre deux agents qui affecte certains objets. Chaque agent et chaque objet ressource peuvent participer à plusieurs actions atomiques. La phrase *"Le client insère la bouteille dans la machine de recyclage"* est un exemple d'action atomique. Cette action implique deux agents "Le client" et "La machine de recyclage". Le paramètre *"la bouteille"* est un objet ressource.

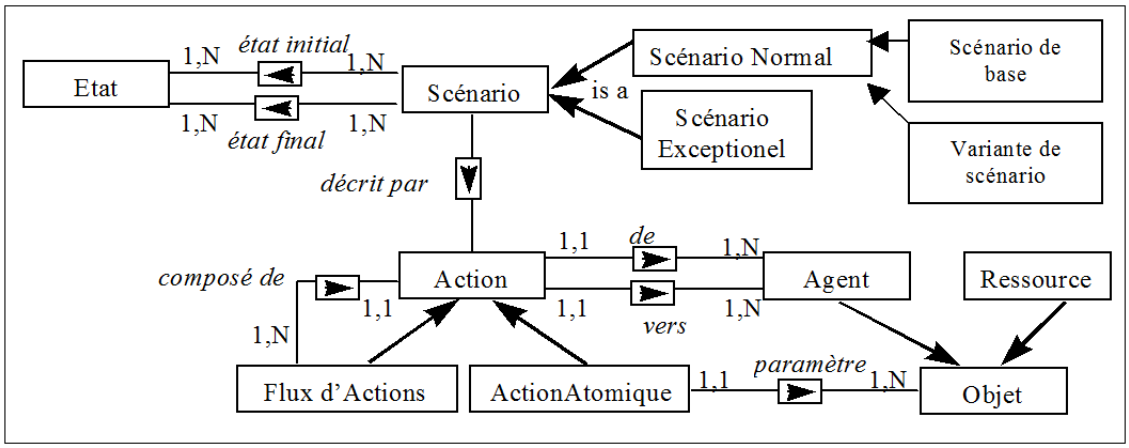


Figure 2.4 — La structure d'un scénario selon CREWS

Donc la phrase peut être exprimée sous la forme suivante :
(le client)$_{agent}$ (insère)$_{verbe}$ (la bouteille)$_{ressource}$ dans (la machine de recyclage)$_{agent}$.

2.4.4 Les approches par points de vue

L'usage de la notion de point de vue (PV) est récurent dans le domaine de l'IE. Ces vues ou points de vues reflètent les compétences, objectifs et rôles de chaque participant dans le processus d'IE. Les points de vue multiples permettent certainement une meilleure élicitation des exigences, et ne peuvent être que bénéfiques au processus d'élicitation des exigences, comme le montrent les travaux de [Nuseibeh et al., 1994], [Sommerville and Sawyer, 1997], [Sommerville, 2001], [Charrel, 2002]. Les points de vue sont utilisés pour guider la découverte, l'analyse et le gestion des exigences. La méthode Preview (Process and Requirements Engineering VIEWpoint) [Sommerville et al., 1998] est l'une des méthodes les plus connues utilisant la notion de point de vue.

2.4.5 La méthode PREview

La notion de point de vue (PV) est définie dans PREview comme une entité qui encapsule quelques-unes des informations qui concernent les exigences d'un système futur. Ces informations peuvent être découvertes à travers un processus d'analyse, à partir des discussions avec les parties prenantes, ou à partir des connaissances organisationnelles et du domaine.

Un PV dans PREview inclut les informations suivantes :
– le nom, utilisé pour identifier le PV.

– le centre d'intérêt du PV (Focus), définit le cadre de définition du PV.
– les préoccupations du PV, reflètent les buts organisationnels et les contraintes qui guident le processus d'analyse.
– Les sources des PVs sont des identifications explicites des sources d'information associées aux points de vue
– les exigences du PV, sont un ensemble de besoins résultant du processus d'analyse du système en se basant sur le centre d'intérêt du PV. Les exigences peuvent être exprimées en terme de fonctions système, les nécessités des utilisateurs ou des contraintes résultantes du domaine de l'application ou à partir des considérations organisationnelles.
– l'historique du PV, celui-ci enregistre les changements du PV comme une aide pour la traçabilité. Il inclut les changements du centre d'intérêt, des sources et des besoins encapsulés dans le PV.

PREview est applicable sur les activités d'élicitation des exigences. Ivan Sommerville [Sommerville et al., 1998] a défini un processus informel pour appliquer l'approche PREview. Le processus de découverte des exigences est divisé en quatre activités, qui sont (cf. fig 2.5) :

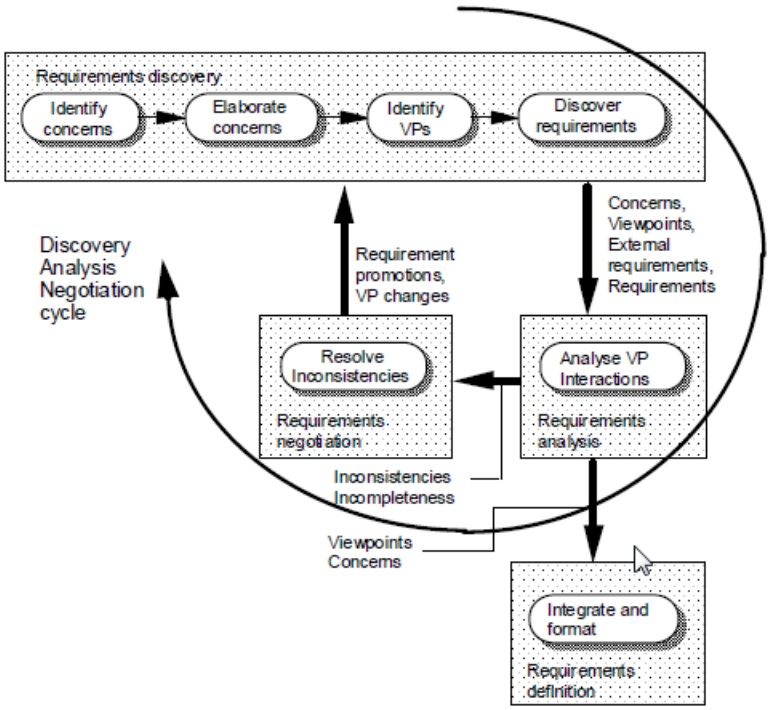


Figure 2.5 — Le modèle du processus PREview [Jethro et al., 2005]

– l'identification des préoccupations (*Identify concerns*).

– l'élaboration des préoccupations *(Elaborate concerns)* comme des exigences externes et des questions.
– l'identification des PVs *(Identify VPs)*
– la découverte *(Discover requirements)* des exigences pour chaque PV,

2.5 Les approches d'ingénierie des exigences orientées Aspect

L'utilisation du paradigme aspect dans la phase d'implémentation du cycle de développement des applications logicielles apporte des solutions aux difficultés du paradigme de l'objet qui sont la dispersion et l'enchevêtrement du code. Un nombre important de langages de programmation orientés aspect sont utilisés (AspectJ, HyperJ, AspectC, JAC,.....) [Jethro et al., 2005].

Mais l'utilisation du paradigme aspect, lors des premières phases du cycle de développement, n'a pas atteint encore un stade de maturité suffisant pour faire face à la complexité des applications. Beaucoup de travaux sont en cours notamment dans la phase de l'IE.

Les techniques d'IE qui reconnaissent de manière explicite l'importance accordée au même titre pour les préoccupations transversales fonctionnelles et non-fonctionnelles ainsi que pour celles de base (non transversales) sont qualifiées d'approches d'IE orientées aspect (Aspect Oriented Requirements Engineering : AORE).

Les trois facteurs suivants sont à l'origine de l'émergence de ces approches [Kiczales et al., 1997].

– Le facteur de composition : l'intégration (assemblage) des exigences séparées.
– Le facteur de traçabilité : suivi des propriétés transversales durant le cycle de vie.
– Le facteur de l'émergence de la programmation orientée aspect (POA).

La section suivante décrit quelques approches particulières dites "Early Aspects Approaches", qui sont : Arcade, ARGM, AOSD/UC et l'approche Theme.

2.5.1 L'approche Arcade

C'est une approche orientée aspect basée sur les points de vue. Cette approche est la première qui a présenté le concept d'aspect lors de l'analyse des exigences. Rashid Awais et al., présentent dans l'article [Awais et al., 2002] pour la première fois le terme d'aspect précoce (Early aspect), ce terme est maintenant reconnu dans toute la communauté de l'IE et de l'architecture du logiciel.

Cette approche propose une technique pour la séparation des exigences aspectuelles et non aspectuelles, ainsi que leurs règles de compositions [Jethro et al., 2005]. La figure 2.6 décrit le modèle général du processus d'IE proposé par cette méthode.

A l'origine cette approche est basée sur la méthode Preview. Elle propose une séparation des exigences orientées aspects et des exigences de base, ces exigences sont fortement indépendantes les unes des autres. Une instanciation concrète de ce modèle est proposée par R. Awais [Awais et al., 2003], en utilisant Preview pour décrire les points de vue et un mécanisme de composition basé sur XML, comme le montre la figure 2.7, qui représente un exemple de point de vue d'Arcade dans un système portugais de paiement de péage [Awais et al., 2003]. Les grandes lignes de cette approche ont été présentées dans [Awais et al., 2003], et développées dans [Kotonya and Sommerville, 1998]. Dans cette méthode,

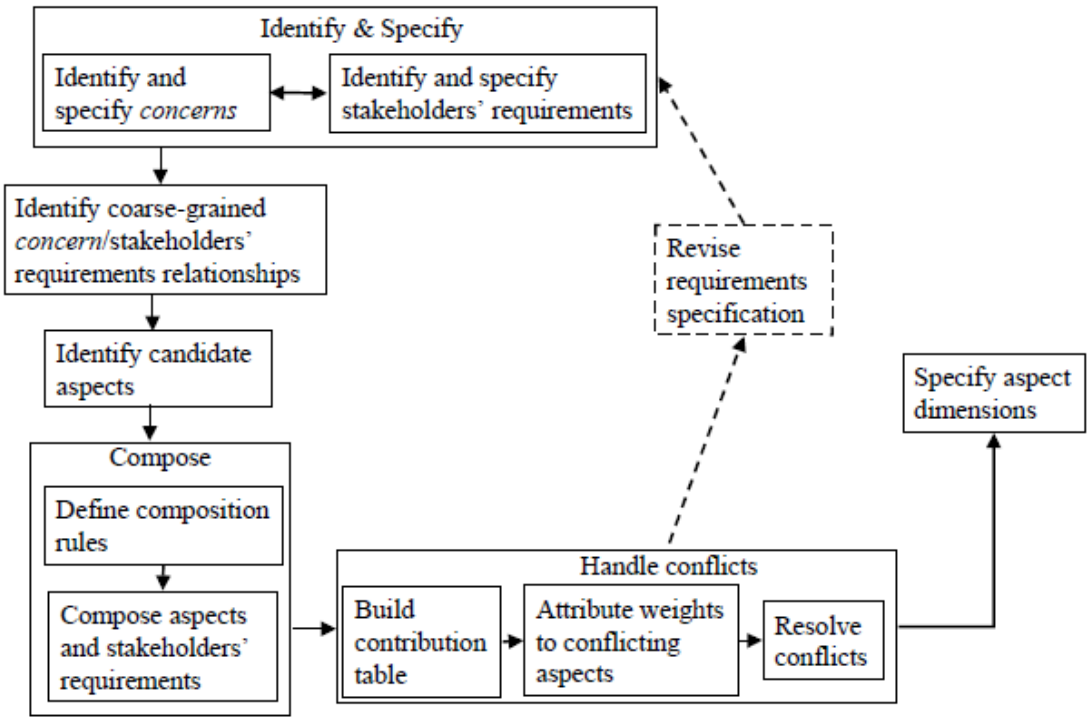


Figure 2.6 — Le processus de l'approche Arcade

les exigences aspectuelles ou aspects sont similaires aux préoccupations de la méthode PREview (PREview concerns) et entrecoupent les exigences des usagers dérivées de plusieurs points de vues. Cependant, l'objectif de la méthode Arcade, est la découverte des exigences et la production d'un document de spécifications de ces exigences, et ceci par la détection et la résolution des situations conflictuelles entre préoccupations, point de

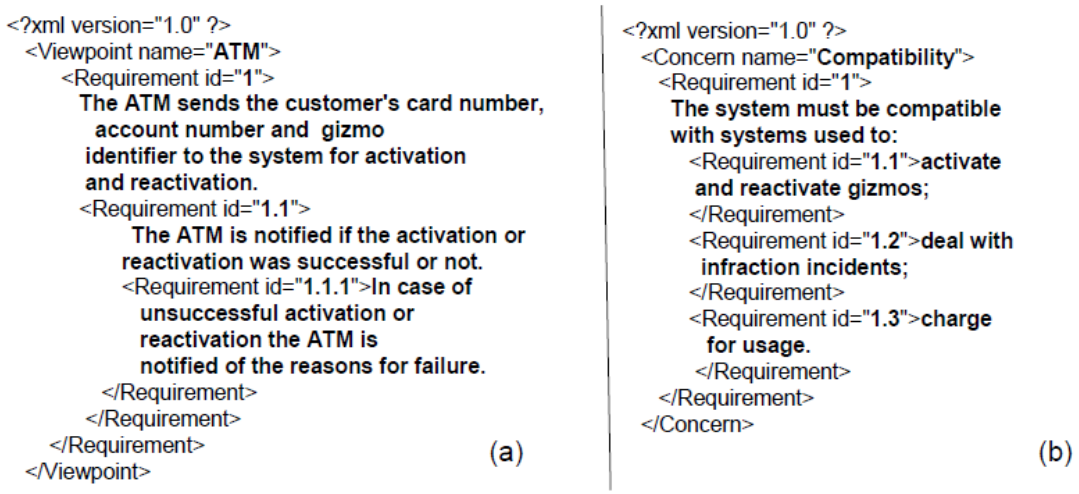


Figure 2.7 — Représentation XML de point de vue et de préoccupation dans AORE with Arcade [Jethro et al., 2005]

vues et exigences.

2.5.2 L'approche ARGM

L'approche ARGM (Aspects in Requirements Goal Models)) est développée par E., Yu [Yu et al., 2004]. Les aspects sont découverts à partir des relations existantes entre les exigences fonctionnelles (`goals`) et les non-fonctionnelles qualifiées de (`softgoals`). ARGM propose un processus pour décomposer les `goals` et les `softgoals`, pour identifier ensuite les aspects. L'article de E. Yu [Yu et al., 2004] présente en détail les différents algorithmes utilisés par ce processus. Selon son auteur, ARGM prévoit un processus récursif pour décomposer les `goals` et les `softgoals` jusqu'à l'aboutissement à des tâches spécifiques. Les exigences transversales (fonctionnelles ou non) peuvent être identifiées comme des tâches qui contribuent à plusieurs `goals` et `softgoals`.

2.5.3 L'approche AOSD/UC

L'approche AOSD/UC (Aspect Oriented Software Development with Use Cases) étend l'approche classique basée sur les cas d'utilisation avec trois principaux concepts qui sont :

– points de coupure (pointcuts) : ce sont des points de jointure représentés par les points d'extension des UC.

– tranche de cas d'utilisation (use cases slice) : qui contient les détails d'un cas d'utilisation à une phase donnée du cycle de développement.
– module de cas d'utilisation (use case module) : regroupe les tranches de cas d'utilisation.

AOSD/UC distingue deux types de cas d'utilisation.
- Cas d'utilisation égaux (peer), ils représentent les exigences de base.
- Cas d'utilisation d'extension (extension) : pour découvrir les exigences entrecoupantes.
Cette approche (AOSD/UC) est utilisée aussi pour capturer les exigences non fonctionnelles comme des cas d'utilisation, en utilisant pour cela le concept d'*infrastructure use case* qui désigne les activités nécessaires utilisées par le système afin de satisfaire les exigences des usagers.

AOSD/UC est une approche qui couvre tout le cycle de développement, de la phase d'IE jusqu'à la phase d'implémentation. Dans la phase d'IE, cette approche propose un processus similaire au processus traditionnel de l'approche des cas d'utilisation. La différence essentielle réside dans la séparation des cas d'utilisation des exigences non fonctionnelles. Cette catégorie d'exigence doit être identifiée, définie, et enregistrée en utilisant le concept d'*infrastructure use cases* avec l'identification et le traitement des cas d'utilisation fonctionnels. Une autre différence est le paquetage de chaque cas d'utilisation avec ces éléments liés dans un seul paquetage dit *use case slice*.

2.5.4 L'approche Theme

Nous présentons une quatrième approche Theme [Baniassad and Siobhán, 2004] qui n'étend pas les approche existantes. L'approche `Theme` couvre les deux phases de développement que sont l'analyse (Theme/Doc) et la conception (Theme/UML). Cette approche considère les aspects comme des préoccupations transverses et réparties dans un système. Theme/Doc traite les aspects lors de la phase de l'IE, elle utilise pour cela un graphe d'analyse.

Dans cette approche, un thème (`Theme`) se définit comme un paquetage paramétré. Les paramètres d'un tel paquetage sont cités dans un rectangle en pointillé dans son coin supérieur droit. Une relation (nommée `bind`) permet d'exprimer la composition paramétrée de deux thèmes. La vérification de la composition est réalisée en ajoutant au graphe d'analyse (Theme/Doc) les informations relatives aux différentes compositions. Les traitements génériques, sont considérés par l'approche comme des fonctionnalités de trace ou de synchronisation. Chaque paramètre correspond à une classe et à l'ensemble de ses méthodes sur lesquelles la fonctionnalité doit être `tissée`. Cette modification du traitement apportée par ce tissage est décrite à l'aide de diagrammes de séquences.

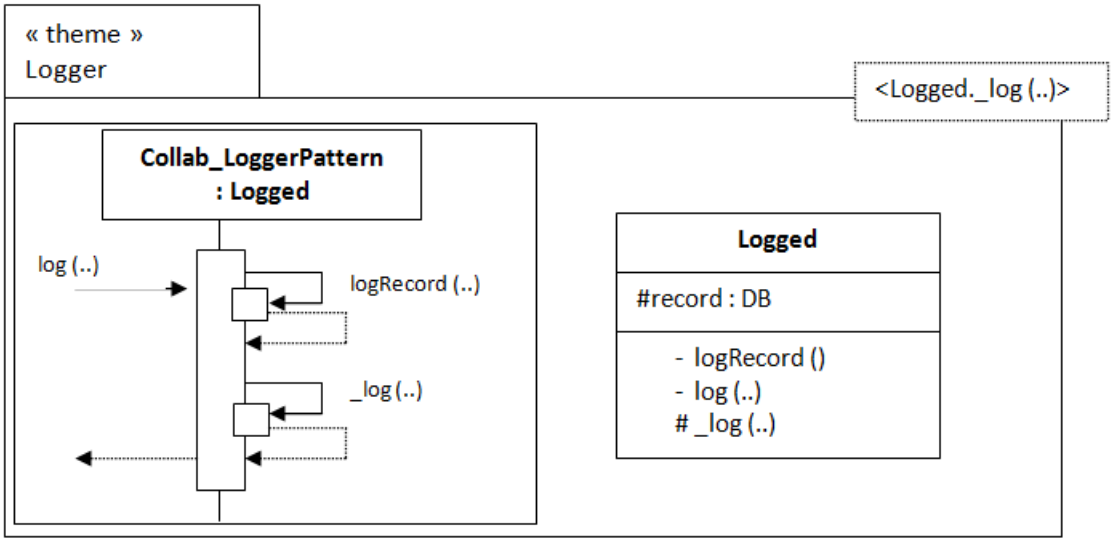


Figure 2.8 — Thème d'une fonctionnalité de trace [Baniassad and Siobhán, 2004]

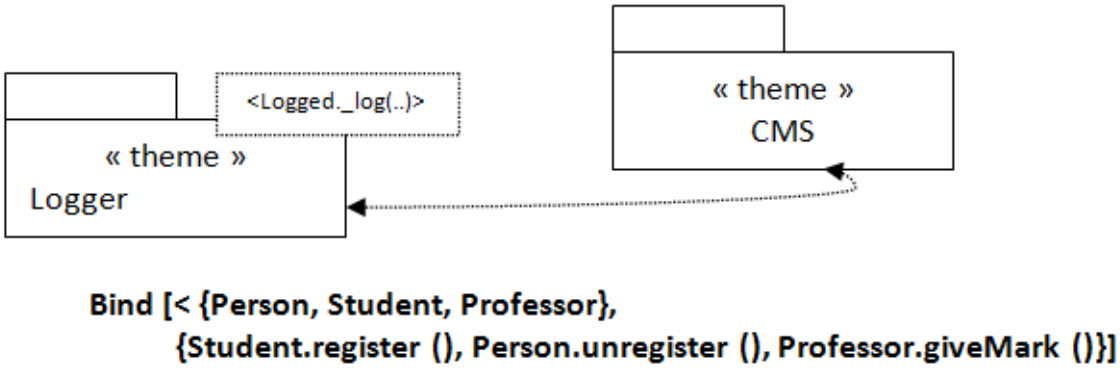


Figure 2.9 — Utilisation du thème de trace [Baniassad and Siobhán, 2004]

La figure 2.8, illustre cela pour la définition d'une fonctionnalité générique de trace. Les paramètres pour ce thème sont ici la classe *Logged* et la méthode *log*. La figure 2.9, illustre l'utilisation de ce thème. La relation **bind** entre le thème *Logger* et le thème *CMS* porte le paramétrage permettant d'appliquer la fonctionnalité de trace aux classes *Person, Student* et *Professor* définies dans le paquetage CMS (Course Management System) à leurs méthodes respectives *register, unregister* et *giveMark*.

2.6 Les approches d'ingénierie des exigences : avantages et inconvénients

Le tableau 2.1 dresse un état récapitulatif des trois grandes catégories d'approche de l'IE à savoir les approches orientées but, scénario et aspect en dressant leurs avantages et inconvénients.

Approche	Avantages	Inconvénients
Approche orientée But	– Elle permet une certaine souplesse dans le choix de méthodes semi-formelles/ formelles pour la modélisation des exigences. – Les buts fournissent la meilleure abstraction pour la décision. – Elle permet une bonne manipulation des conflits.	– Les techniques et les pratiques en rapport avec l'étape d'élicitation sont relativement faibles. – Nécessite une période importante de temps pour la compréhension et la maîtrise. – Elle souffre de l'absence des outils de modélisation fiables et des supports automatisés et logiciels. – Ne prend pas en compte le management des exigences.

Approche	Avantages	Inconvénients
Approche orientée Scénario	– Assistance logicielle assez importante (CREWS-SAVER, RequisitePro, L'Ecritoire). – Fournit un langage bien défini pour décrire les scénarios et spécifier les exigences. – Utilisation des modèles de besoins génériques. – Elle permet la réutilisation extensive via les modèles de décisions et les besoins génériques. – Elle utilise des techniques de génération de scénarios automatiquement. – Elle permet un traitement des cas exceptionnels. – Elle possède un bon moyen de validation semi-automatique des besoins qui se base sur les patrons de reconnaissance de comportement.	– La difficulté de l'assurance de complétude de la spécification des exigences à cause de la fragmentation des exigences saisies dans de multiples scénarios . – Elle rend également difficile la recherche des différents aspects d'une même fonctionnalité du système à travers de multiples scénarios. – Le caractère "ad hoc" du processus d'identification des variations du scénario de base d'un cas d'utilisation. – Des difficultés rencontrées dans la maitrise des niveaux d'abstraction, un même scénario pouvant comporter des actions de différents niveaux d'abstraction et donc faisant référence à des exigences de différents niveaux .

Approche	Avantages	Inconvénients
Approche orientée Aspect	– Réduit la complexité. – Met en valeur la compréhensibilité et la traçabilité à travers tout le processus de développement. – Permet l'encapsulation des préoccupations et des besoins non fonctionnels (besoins de qualité). – Elle bénéficie des avantages des préoccupations Crosscutting. – Elle fournit de bons moyens d'identification, séparation, représentation et composition des préoccupations dans des étapes plus tôt dans le processus de développement. – Permet une bonne maîtrise des situations conflictuelles.	– Son support logiciel est plutôt maigre. – Le langage de définitions des règles de composition est incomplet. – Les méthodes proposées ne suivent pas un chemin purement systématique. – Les méthodes proposées ne prennent pas en considérations toutes les situations conflictuelles émergentes. – Les études sur le domaine, et l'influence du paradigme d'aspect sur le cycle de vie sont maigres.

Tableau 2.1: Tableau comparatif des différentes approches d'IE .

2.7 Conclusion

Le but de ce chapitre était de faire un état de l'art de la discipline de l'Ingénierie des Exigences (IE) de manière générale, en mettant l'accent en particulier, sur les approches d'IE orientées aspect (AORE en anglais). Il vise à présenter le cadre de nos travaux à savoir une approche orientée aspect d'IE dans un environnement multi entreprise.

Nous avons abordé la discipline de l'IE en présentant ses principales étapes. De ce fait, ce chapitre est structuré en deux grandes parties, la première visait à définir le domaine de l'IE avec tous les aspects liés à ce domaine. La deuxième partie se destinait à faire un aperçu sur les approches de l'IE. Ces approches ont été divisées en deux classes, celles des approches dites classiques, et celles orientées Aspects.

Les approches classiques d'IE sont les approches orientées But, Scénario et hybrides (but et scénario), et basées point de vue. Pour chaque type d'approche un exemple est présenté. Les approches orientées aspects sont celles qui soulignent de manière forte l'importance accordée aux préoccupations entrecoupantes fonctionnelles et non-

fonctionnelles ainsi qu'à celles de base (non entrecoupantes). Ces approches sont connues par l'acronyme (AORE : Aspect oriented requirements engineering).

Le paradigme aspect a apporté un moyen efficace permettant une meilleure réutilisation des artefacts. Dans cet esprit, notre objectif est de proposer une approche aspect couvrant les deux premières phases d'analyse et de conception pour le développement d'un Système d'information coopératif en réutilisant des artefacts des systèmes existants.

Le chapitre suivant porte sur l'Ingénierie Dirigée par les Modèles (IDM) et les principales approches de la modélisation orientée aspect.

3 La composition de modèles dans l'Ingénierie Dirigée par les Modèles

Il importe en peinture, que le portrait ressemble au modèle, mais non pas le modèle au portrait (Paul-Jean Toulet).

3.1 Introduction

Afin de gérer la complexité sans cesse croissante des systèmes, en particulier les systèmes logiciels, les modèles sont présents et largement utilisés, depuis bien longtemps dans leur processus de développement. Le développement des outils informatiques a fait évoluer les modèles depuis les formats papier ou les prototypes de taille réduite vers des formats électroniques de ces modèles. Dans le cycle de développement des systèmes, les modèles ont pris un rôle important depuis les phases d'expression des exigences jusqu'aux phases d'exploitation ou de maintenance, grâce aux outils de construction et d'exploitation des modèles qui ont été développés.

La modélisation permet de séparer des préoccupations en faisant abstraction des aspects spécifiques de la réalité pour des objectifs précis [Favre et al., 2006]. Cependant, les modèles jouent un rôle de tout premier plan dans l'Ingénierie Dirigée par les Modèles (IDM), ou (Model Driven Engineering, MDE) [Stuart, 2002], [Favre et al., 2006]. Lors des phases d'analyse et de conception dans le développement logiciel l'approche IDM s'est largement répandue ces dernières années. La nouveauté de l'IDM réside dans l'utilisation systématique de métamodèles. Cette approche se diffère des méthodes de modélisation traditionnelles comme Merise par la préoccupation constante de rendre les (meta)modèles productifs. C'est à dire de rendre les modèles interprétables et manipulables par la machine. Pour l'IDM les aspects, tels qu'on les a définis dans les chapitres précédents, sont pris en compte par un modèle particulier conforme à un métamodèle.

En résumé, la séparation des préoccupations est gérée à tous les niveaux d'abstraction avec l'utilisation de métamodèles dans l'approche IDM.

L'IDM cible une production logicielle bien fondée, en prenant en charge la complexité et l'évolution croissante des applications. Cependant, l'IDM ouvre de nouvelles voies d'investigation [Favre et al., 2006]. Ce chapitre a pour ambition (i) de donner une vision rapide des principaux concepts de l'IDM, en présentant les concepts de base, (ii) de présenter quelques approches de composition de modèles.

3.2 L'ingénierie dirigée par les modèles : concepts de base

L'IDM est déployée pour intégrer, prolonger et renforcer les approches de développement déjà existantes. Dans cette approche les concepts de modèle, de langage, de métamodèle et de transformations sont employés. La section suivante les présente en détail.

3.2.1 Les modèles

Il n'existe pas de définition universelle du mot "modèle" (tout comme il n'existe pas une définition unique pour : objet, aspect, langage, etc. [Favre et al., 2006]), mais nous essayons de donner quelques définitions existantes dans la littérature : la première définition provient du standard UML.

> **définition 1 :** *a model is an abstraction of a physical system, with a certain purpose. [Seidewitz, 2003]* . C'est à dire : "un modèle est une abstraction d'un système physique, dans un but donné".

> **définition 2 :** *a model is set of statements about some system under study.*

Bézivin et Gérbé donnent la définition suivante :

> **définition 3 :** *a model is a simplification of a system built with an intended goal in mind* [Bézivin and Gerbé, 2001]. C'est à dire un modèle est une simplification d'un système construit dans une intention particulière.

Une autre définition donnée par Kleppe et al. :

> **définition 4 :** *A model is a description of (part of) a system written in well-defined language* [Kleppe et al., 2003]. C'est à dire un modèle est une description d'une partie d'un système écrit dans un langage bien défini.

3.2.2 Les métamodèles

La manipulation d'un modèle par une machine le rend productif, [Favre et al., 2006]. De ce fait, le langage dans lequel ce modèle est exprimé doit être clairement défini, ce langage représente un métamodèle, d'où la définition de l'OMG (Object Management Group) suivante du métamodèle :

> *a metamodel is a model that defines the language for expressing a model.*

Selon Kleppe [Kleppe et al., 2003], un métamodèle est défini comme suit :

> *a metamodel is a specification model for a class of SUS*[1] *where each SUS in the class is itself a valid model expressed in a certain modelling language.*

Donc, un métamodèle est un modèle d'un langage de modélisation, ce qui définit une relation nommée *ConformeA* entre un modèle et son métamodèle [Bézivin, 2004].

La Figure 3.1 présente les quatre couches de l'architecture de modélisation telle que définie par l'OMG [Bézivin, 2004].

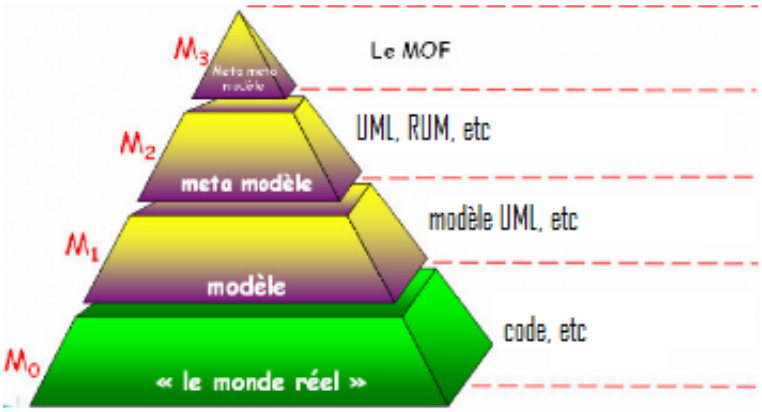


Figure 3.1 — Pyramide de modélisation de l'OMG

1. Le niveau M0 représente le monde réel : par exemple, une application qui s'exécute. Il contient les informations du monde réel que l'on souhaite modéliser.
2. Le niveau M1 est composé des modèles. Les informations de M0 sont décrites par un modèle appartenant au niveau M1. Par exemple, un modèle UML appartient au niveau M1.
3. Le niveau M2 représente les langages de définition de modèles. Ces langages sont les métamodèles. Par exemple, le métamodèle UML permet de définir des modèles (niveau M1).

1. SUS : System Under Study

4. Le niveau M3 est composé du langage unique de définition des métamodèles, appelé MOF (Méta Object Facility). C'est un métamétamodèle qui définit la structure de tous les métamodèles qui se trouvent au niveau M2. Le MOF est réflexif, c'est-à-dire qu'il s'auto-décrit, ce qui permet de dire que le niveau M3 est le dernier niveau de la hiérarchie. Le niveau M3 correspond donc aux fonctionnalités universelles de modélisation logicielle, alors que le niveau M2 correspond aux fonctions spécifiques des différentes familles, chaque fonction étant prise en compte par un métamodèle spécifique.

3.2.3 Les transformations de modèles

La transformation de modèle est un concept clé dans l'IDM. Elle consiste à pouvoir rendre productifs (opérationnels) les modèles. En effet, l'intérêt de transformer un modèle *Ma* en un modèle *Mb* que les métamodèles respectifs *MMa* et *MMb* soient identiques (transformation endogène) ou différents (transformation exogène) apparaît comme primordial pour réaliser des opérations telles que la génération de code, le refactoring, la migration technologique, etc. [Bézivin, 2004].

Autrement dit, une transformation de modèles se définit par l'opération de génération d'un ou plusieurs modèles cible à partir d'un ou de plusieurs modèles source, conformément à une définition de transformation [Bézivin, 2004]. Dans la littérature trois types de transformation de modèles sont définies, à savoir [Anwar, 2009] :

1. les transformations simples (1 vers 1) : pour chaque élément du modèle source est associé un seul élément du modèle cible.
2. les transformations multiples (M vers N) qui associent à un ensemble d'éléments du modèle source un ensemble d'éléments (généralement différents) du modèle cible (par exemple, la fusion de modèles est un exemple de transformation M vers 1).
3. les transformations de mise à jour appelées aussi modifications sur place (ajout, suppression, changement de propriétés, etc.). Ces transformations sont caractérisées par l'absence de modèle cible, et agissent donc directement sur le modèle source. Les transformations de restructuration (refactoring) sont des exemples typiques de ce type de transformation.

3.3 L'approche MDA

L'OMG (Object Management Group) a proposé l'approche Model Driven Architecture (MDA). MDA est une concrétisation de l'approche IDM. Elle se base sur la séparation des spécifications fonctionnelles d'un système, de son implémentation sur une plate-forme spécifique [Favre et al., 2006]. Cette approche est basée sur le principe de la

séparation des préoccupations. En partant de la phase de l'IE jusqu'à la phase d'implémentation, selon le processus de développement orienté MDA, tout est considéré comme modèle.

MDA identifie quatre types de modèles : `CIM`, `PIM`, `PDM` et `PSM`.

1. Le `CIM` (Computation Independent Model), modèle de domaine ou modèle métier, modélise les exigences du système. Il sert à la compréhension du problème et à fixer un vocabulaire commun pour un domaine particulier [Djebbi and Gervais, 2003/2004]. Les exigences exprimées dans le CIM doivent être tracées dans les modèles qui seront élaborés dans les autres phases du cycle de développement.
2. Le `PIM` (Platform Independent Model) décrit le système indépendamment de détails de plate-forme particulière. Plusieurs modèles `PIM` peuvent être élaborés. Ces modèles représentent les fonctionnalités métiers. Les `PIM` peuvent contenir des aspects technologiques et architecturaux mais sans détails spécifiques d'une plate-forme. Un `PIM` doit être raffiné en rajoutant les détails spécifiques aux plate-formes pour produire un `PSM`.
3. Le `PDM` (Platform Description Model) décrit la plate-forme sur laquelle le système va être exécuté (par exemple des modèles de composants à différents niveaux d'abstraction comme CCM (CORBA Component Model) et EJB (Entreprice Java Beans)).
4. Le `PSM` (Platform Specific Model) est le résultat de la transformation du modèle PIM prenant en compte les spécificités de la plateforme d'exécution.

3.4 La programmation par aspect et l'IDM

Selon J.M. Favre [Favre et al., 2006], les approches comme la programmation par aspect cherchent à réaliser la synthèse des préoccupations logicielles sur la base du code, par exemple avec AspectJ ou HyperJ. Dans l'approche MDA on essaie de réaliser la séparation des préoccupations ou aspects liés ou indépendants de la plateforme par des modèles différents (PIM et PSM). Dans les approches d'IDM, chaque aspect ou point de vue d'un projet de développement ou de maintenance est pris en compte par un modèle particulier conforme à un métamodèle.

La séparation et le tissage des aspects sont bien présents dans les approches IDM. Cependant, la séparation des aspects liés ou indépendants de la plateforme est jugée prioritaire dans le MDA, mais pour l'IDM les aspects à prendre en compte touchent toutes les préoccupations qui peuvent apparaitre lors de tout le cycle de développement. De façon générale, la programmation par aspect gère la séparation des préoccupations au niveau du code, par contre l'IDM gère la séparation des préoccupations à tous les niveaux d'abstraction.

3.5 La composition de modèles

Dans les processus de développement d'IDM, un concept important est celui de la composition de modèles, les motivations pour cela sont multiples. La composition apparaît sous divers termes selon le contexte d'application. De façon générale, il s'agit d'une opération qui consiste à combiner un nombre de modèles pour en produire un ou plusieurs.

A titre d'exemple, en IE basée point de vue, les exigences du système sont exprimées selon plusieurs points de vue. Il en résulte un ensemble de modèles appelés *vues*. Cependant, la fusion des vues partielles génère un modèle unique et partageable, basé sur les dépendances entre les vues, il gère la cohérence du système en résolvant les éventuelles incohérences entre les vues [Sabetzadeh and Easterbrook, 2005].

On retrouve la composition de modèle dans plusieurs disciplines telles que les bases de données [Elmasri and Navathe, 1999]. Dans ce domaine la composition de modèles est connue sous le nom d'*intégration*, en ingénierie des connaissances sous le nom *d'alignement d'ontologies* [Ehrig et al., 2005]. On peut signaler aussi le travail de S. Mosser et al. [Mosser et al., 2008] relatif à la composition de services et d'activités logicielles.

Dans le contexte du paradigme aspect ou le développement de logiciels par aspects, la composition de modèles est connue par le terme de *Tissage* ou *Weaving*. Le tissage consiste à composer des modèles d'aspects représentant des préoccupations transversales avec le modèle de base. Dans notre travail, nous nous intéressons particulièrement à ce type de composition de modèle.

Les techniques de modélisation par aspects considèrent les aspects dans les modèles (ie : au niveau conception). Elles découlent essentiellement de celles de la programmation par aspects. Dans cette perspective, on retrouve deux catégorie d'approches. La première catégorie regroupe les approches visant à modéliser les programmes aspects [Basch and Sanchez, 2003], [Barra et al., 2004]. Elles proposent donc des constructions permettant de faire apparaître dans un modèle objet les éléments propres à la programmation par aspects comme les points de jonction et les greffons. Ces approches s'intéressent en particulier à représenter des détails techniques pour la mise en œuvre des modèles dans un langage orienté aspect [Barra et al., 2004], [Cottenier et al., 2007]. Contrairement à ces approches, les approches de la seconde catégorie proposent, par contre, de structurer les aspects au niveau des modèles. Cette catégorie d'approche cherche à promouvoir le paradigme aspect à des niveaux d'abstraction supérieurs. Elle propose des méthodes de développement logiciel orientées aspect, couvrant tout le cycle de vie, allant de la phase d'IE jusqu'à l'implémentation. C'est à cette seconde famille que nous nous intéressons dans notre travail. La suite de cette section fait le tour de quelques approches citées dans la littérature. Nous décrivons ainsi les approches de Clarke [Siobhán, 2002], [Baniassad

and Siobhán, 2004], et de France et al. [France et al., 2004], [Raghu et al., 2006], de Muller et al. [Muller, 2006] et de Del Fabro et al. [DelFabro et al., 2006].

3.5.1 La composition de modèles dans PackageMerge d'UML

PackageMerge permet d'étendre les concepts définis dans un paquetage par des fonctionnalités définies dans un autre [Soley and the OMG Staff Strategy Group, 2000]. Cette notion est apparue dans UML2.0. Le concept de combinaison apporté par *PackageMerge* est similaire au concept de *généralisation* qui permet à un élément fils de rajouter les caractéristiques de l'élément père à ses propres caractéristiques. Ce mécanisme est appelé dans le cas où les éléments définis dans différents paquetages à fusionner ont le même nom et représentent le même concept. Il est utilisé le plus souvent pour fournir des définitions différentes d'un même concept destiné à être utilisé différemment, à partir d'une définition commune de base. La relation *PackageMerge* diffère de l'importation de paquetage définit par le concept de relation *include*. Conceptuellement, *PackageMerge* est considéré comme une opération qui combine le contenu de deux paquetages pour produire un nouveau paquetage.

Pour illustrer l'utilisation de *PackageMerge*, nous reprenons l'exemple de [Zito et al., 2006]. Dans cet exemple on cherche à fusionner deux paquetages *EmployeeFunction* et *EmployeeLocation* comme le montre la figure 3.2. Par le biais de la relation *merge*, la définition du concept *Employee* du paquetage *EmployeeFunction* est étendue par une information indiquant le département auquel il est rattaché.

Dans cet exemple, le paquetage *EmployeeFunction* constitue la source de la relation *PackageMerge*, et le paquetage *EmployeeLocation* joue le rôle de paquetage cible (récepteur). Le résultat de la fusion de ces deux paquetages donne lieu au paquetage *EmployeeFunction* enrichi des informations du paquetage *EmployeeLocation* comme le montre la figure 3.3.

En effet, les deux classes *Employee* des deux paquetages sont fusionnées en ajoutant l'attribut *codeEmployee* et l'opération *moveTo()* à la classe *Employee* du paquetage *EmployeeFunction*. La classe *Department* et l'association *worksAt* n'ont pas de correspondant et sont copiées dans le paquetage résultant.

3.5.2 La composition de modèles dans l'approche Theme/UML

Theme [Baniassad and Siobhán, 2004] est une approche d'analyse et de conception orientée aspect, développée essentiellement pour supporter des préoccupations transverses (transactions, distribution, etc.). L'approche Theme propose un modèle de développement à deux niveaux : le niveau d'identification des exigences (Theme/Doc) et le niveau de conception (Theme/UML).

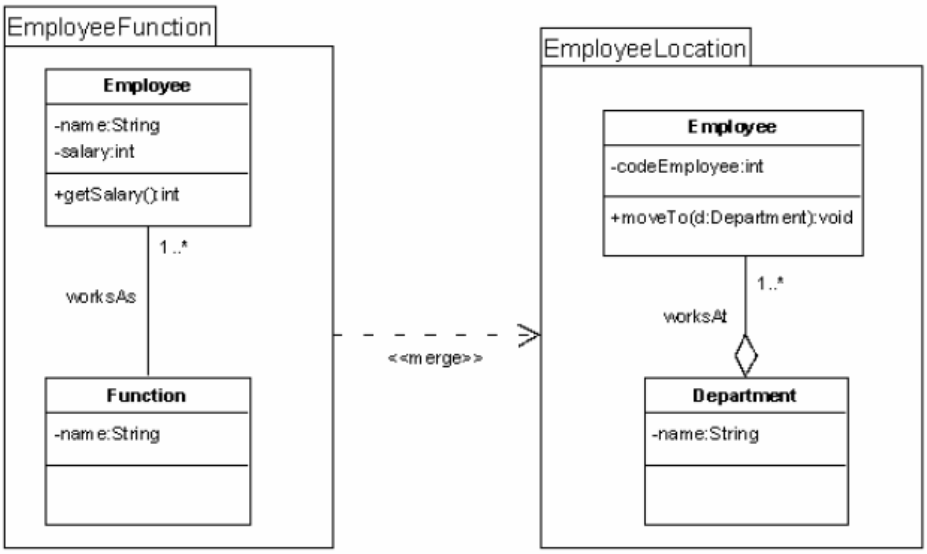


Figure 3.2 — Exemple de PackageMerge [Zito et al., 2006]

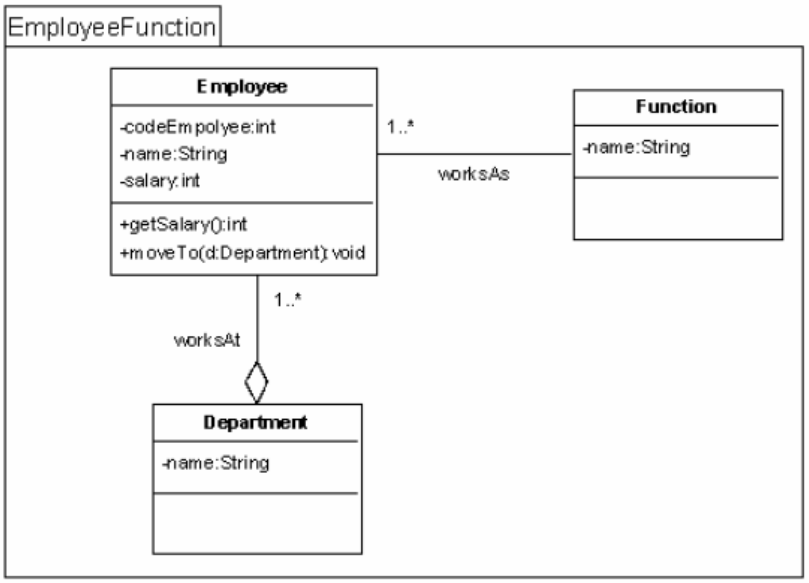


Figure 3.3 — Résultat de PackageMerge [Zito et al., 2006]

Comme on l'a présenté dans le section 2.6.4 du chapitre 2, un thème *(Theme)* se définit comme un paquetage paramétré. Dans Theme/UML, la composition de modèles est spécifiée par une relation de composition qui identifie les parties identiques dans les thèmes à composer, tout en spécifiant comment elles doivent être composées. Cette approche définit trois types de composition : (i) la fusion (*merge*), (ii) la substitution (*bind*), (iii) le remplacement (*override*).

La fusion est utilisée pour la composition de deux thèmes de base. La substitution est surtout utilisée pour la composition d'un thème de base avec un thème d'aspect. La relation de remplacement est utilisée pour remplacer le comportement décrit par un thème par un autre comportement décrit par un autre thème. Chaque relation de composition est associée à une stratégie d'intégration. Par exemple, dans le cas de la fusion, cette relation spécifie comment deux modèles ayant des concepts correspondants sont fusionnés.

Pour la composition de modèles, Theme/UML étend le métamodèle d'UML. Theme/UML définit le concept d'élément composable qui désigne tout élément participant à une relation de composition. Les éléments composables sont divisés en deux catégories : primitifs (attributs, opérations, etc.), et composites (classes, paquetages, etc.) illustrés dans la figure 3.4. Les éléments primitifs ont la particularité d'être composés avec d'autres éléments primitifs tout en gardant leurs spécifications. Par contre, en ce qui concerne les éléments composites, leurs composants ne sont pas considérés comme éléments faisant partie intégrale du composé, et sont par conséquent examinés séparément pendant la composition.

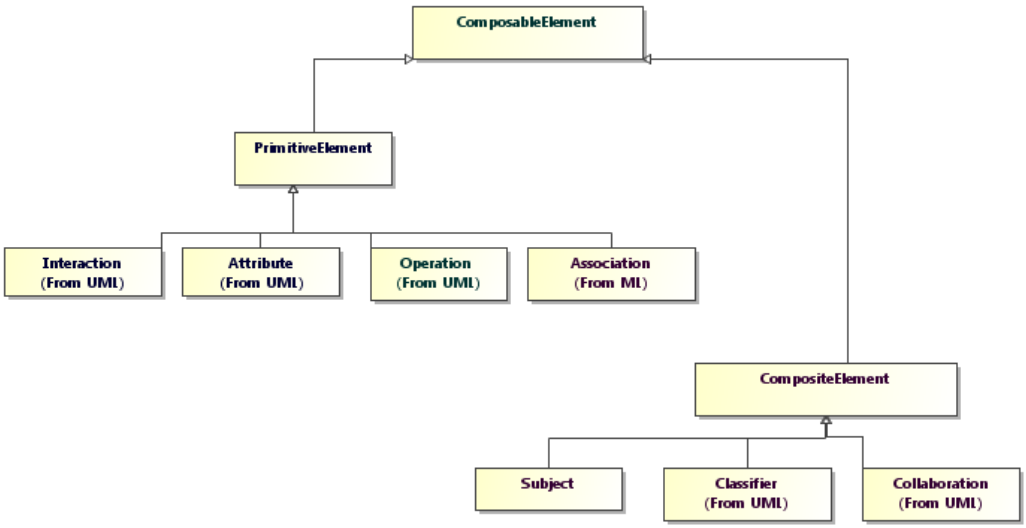


Figure 3.4 — Le métamodèle de Theme/UML

Le concept de thème, apporté par l'approche Theme/UML, offre plusieurs avantages tels que :

1. la décomposition du modèle de conception en thèmes peut faciliter la compréhension du système.
2. il peut résoudre les problèmes de conception tels que la dispersion de la modélisation d'une exigence dans plusieurs éléments de modélisation, ainsi que l'enchevêtrement de plusieurs exigences dans une même unité de conception.

Toutefois, le manque d'outils et de formalisation des mécanismes est à signaler. En plus Theme/UML n'étudie pas les problèmes de l'ordre de composition ni la possibilité de composer les fonctionnalités génériques entre elles. Cette composition est importante pour l'évolutivité et le passage à l'échelle.

3.5.3 La composition de modèles dans l'approche de France et al.

Dans le contexte de la modélisation par aspects, France et al. [France et al., 2004] ont proposé l'approche AAM (Aspect Architecture Modelling). Elle est utilisée pour le tissage d'aspects, qui modélisent les préoccupations transversales d'une application future avec un modèle de base appelé *modèle primaire*. Le tissage de modèles selon AAM repose sur un algorithme de composition et un ensemble de règles de composition appelées *directives de composition* [Raghu et al., 2006].

Les *directives de composition* permettent d'effectuer des opérations, sur les éléments des modèles, comme la création/suppression d'un élément, la modification de la valeur d'une propriété, et sur les modèles d'aspects, comme la résolution de conflits inter aspects par la définition de l'ordre de composition des aspects. Cette approche propose un métamodèle de composition. Ce métamodèle est une extension du métamodèle d'UML. Il ajoute les spécifications des comportements de composition et des méta-opérations qui implémentent la comparaison entre les éléments en se basant sur leurs signatures. La signature d'un élément du modèle comprend les valeurs liées à ses propriétés. A titre d'exemple, si les propriétés d'une classe UML comprennent les propriétés *nom* et *estAbstraite*, alors la signature d'une classe concrète ayant comme nom *étudiant* est définie par { *nom=étudiant, estAbstraite = faux*}.

Selon cette approche, un modèle d'aspect se définit comme un patron caractérisant une famille de solutions logiques de préoccupations. Les patrons sont décrits par des paquetages paramétrés inspirés d'UML2 en utilisant une adaptation d'un langage de description de patrons basé sur UML appelé RBML (Rule-Based Metamodelling Language) [France et al., 2004].

Pour décrire le modèle d'architecture de l'application, l'approche AAM manipule trois types de modèles :

– le *modèle primaire* qui représente le cœur fonctionnel de l'application ;
– les *modèles d'aspects* qui représentent les préoccupations transversales ; ainsi que les associations (*bindings*) utilisées pour instancier les aspects selon le contexte de l'application ;
– les directives de composition qui expriment comment composer les modèles d'aspect instanciés avec le modèle primaire afin de produire le modèle composé de l'application.

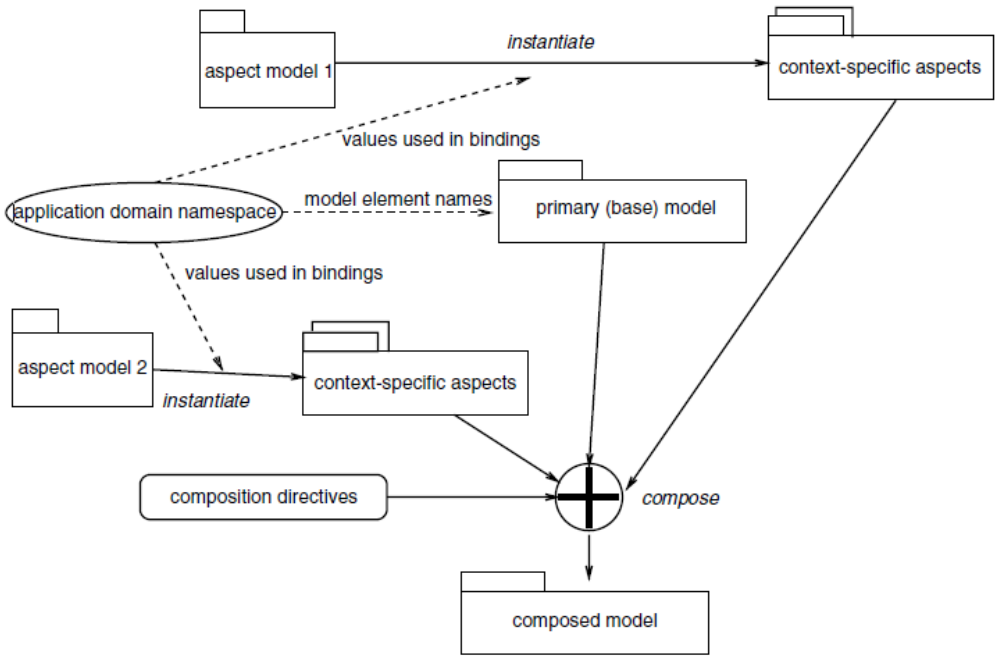


Figure 3.5 — Processus de composition des modèles dans l'approche AAM

Le tissage de *modèles d'aspects* avec le modèle primaire doit être précédé de l'instanciation d'aspects. Cette opération d'instanciation est faite en substituant les paramètres définis dans l'aspect générique par les valeurs spécifiques de l'application. Le modèle d'aspect instancié est appelé modèle d'aspect spécifique à un contexte. La Figure 3.5 présente le processus global de tissage selon l'approche AAM.

Les directives de composition sont les *directives d'éléments* et les *directives de modèles*. Les *directives d'éléments* sont utilisées pour renommer, créer, supprimer et ajouter des éléments à un modèle. Les *directives de modèles* sont utilisées pour déterminer l'ordre de composition des modèles d'aspect. Les *directives de modèles* sont classées en deux types :

1. les directives de pré-Fusion : elles spécifient des modifications sur les modèles avant qu'ils ne soient fusionnés. Ces changements peuvent forcer ou rejeter les correspon-

dances entre les éléments de modèles. Par exemple, dans le cas de deux classes qui représentent le même concept et qui apparaissent dans deux modèles avec deux noms différents, une directive de pré-fusion de type *Rename* est appliquée dans un modèle pour changer le nom de la classe de telle sorte qu'elle corresponde à la classe de l'autre modèle.

2. les directives de post-Fusion : elles spécifient des modifications sur le modèle composé. Par exemple, un aspect de sécurité peut exiger la suppression des associations présentes dans les autres aspects. Cette restriction est exprimée par une directive de post-fusion qui retire ces associations du modèle composé.

L'exemple des figures 3.6 et 3.7 illustre l'utilisation de l'approche AAM [Jackson and Siobhán, 2006] pour la composition d'un modèle d'aspect, représenté par le paquetage *CloseAuction* (cf. fig. 3.6), avec le modèle primaire *OfferSlice* (cf. fig. 3.7). Le modèle d'aspect générique stéréotypé *Context Free Aspect* doit être instancié pour devenir un aspect spécifique avant qu'il puisse être fusionné avec le modèle primaire. L'opération d'instanciation exige l'attachement des éléments appartenant au modèle primaire aux paramètres définis dans le modèle d'aspect générique (l'aspect *CloseAuction* est défini comme un paquetage paramétré). Une instanciation est définie par une association *bind* qui décrit les substitutions appariant les paramètres (préfixés par le symbole syntaxique (|)) définis dans le modèle d'aspect *CloseAuction*. Une fois le modèle d'aspect instancié, on peut effectuer la fusion (*Merge*). Les concepts représentés par des classes sont fusionnés de façon à ce qu'ils apparaissent une seule fois dans le modèle composé résultant (cf. fig. 3.8).

3.5.3.1 Le métamodèle de l'approche de France et al.

Le figure 3.9 présente le métamodèle de l'approche AAM. La métaclasse abstraite `Mergeable` caractérise les éléments de modèle qui peuvent être fusionnés (Property, Classifier, Model, etc.). Elle contient les opérations utilisées pour la fusion des éléments de même type. Le méta-attribut `sigType` représente le type de signature qui détermine l'instance de la signature associée à une instance de `Mergeable`. Une instance de la métaclasse `Composer` se charge de la fusion de deux modèles. Ceci s'effectue par l'opération `mergeModels` qui compare les signatures des deux modèles à fusionner et produit un nouveau modèle lorsque les deux signatures correspondent. Une instanciation du métamodèle de composition inclut un ensemble de modèles, dans lesquels chaque élément d'un modèle est associé à une classe `Signature` qui décrit la signature de l'élément. Une instanciation se compose aussi d'une instance de la métaclasse `CompositionManager`. La métaclasse `CompositionManager` gère la fusion de tous les modèles impliqués dans le modèle de conception de l'approche AAM. La méta-opération `mergeAllModels` crée une instance de `Composer`, lui associe deux modèles, puis invoque l'opération `mergeModels` décrite

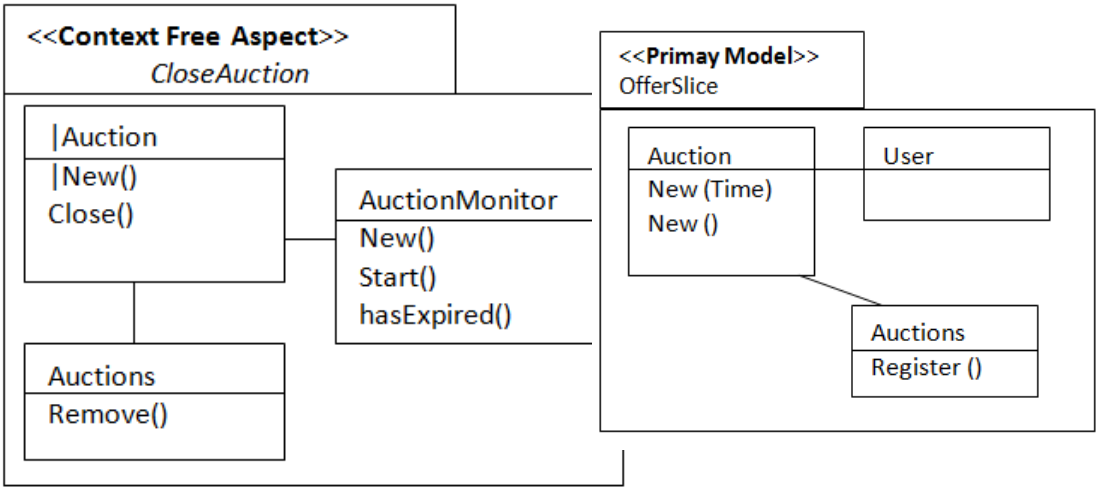


Figure 3.7 — Modèle primaire

Figure 3.6 — Modèle d'aspect générique *CloseAuction*

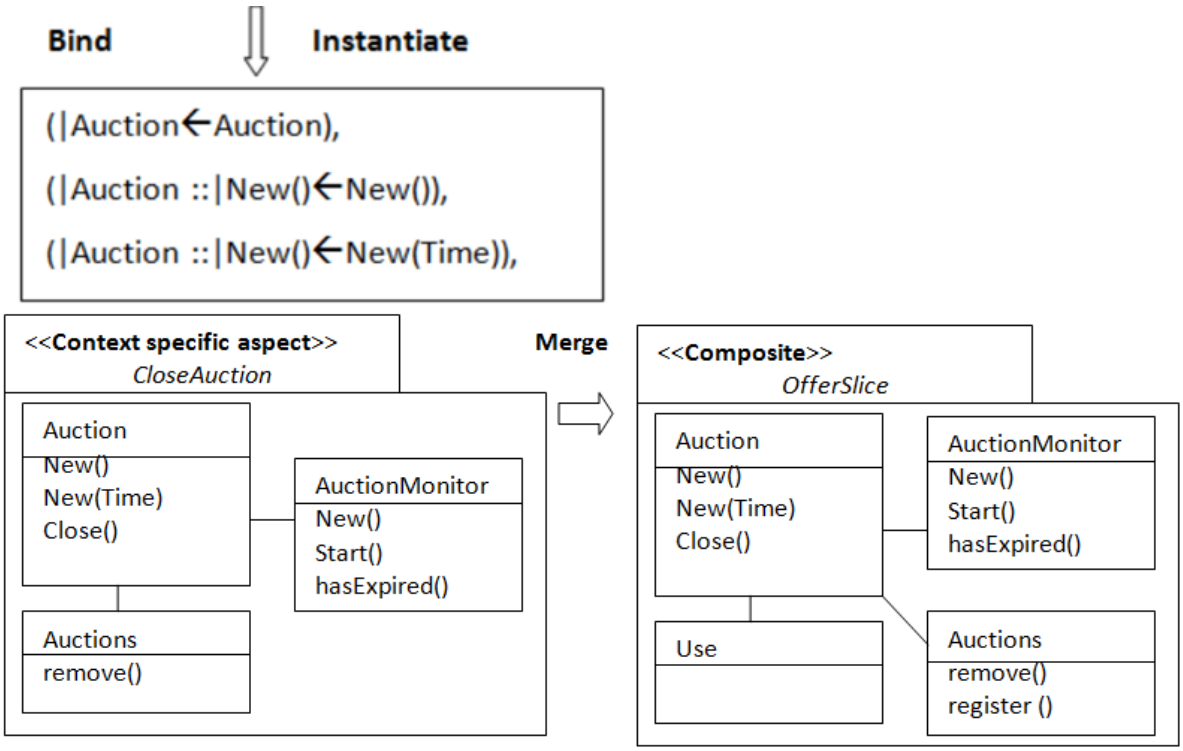


Figure 3.8 — Modèle résultat de la composition

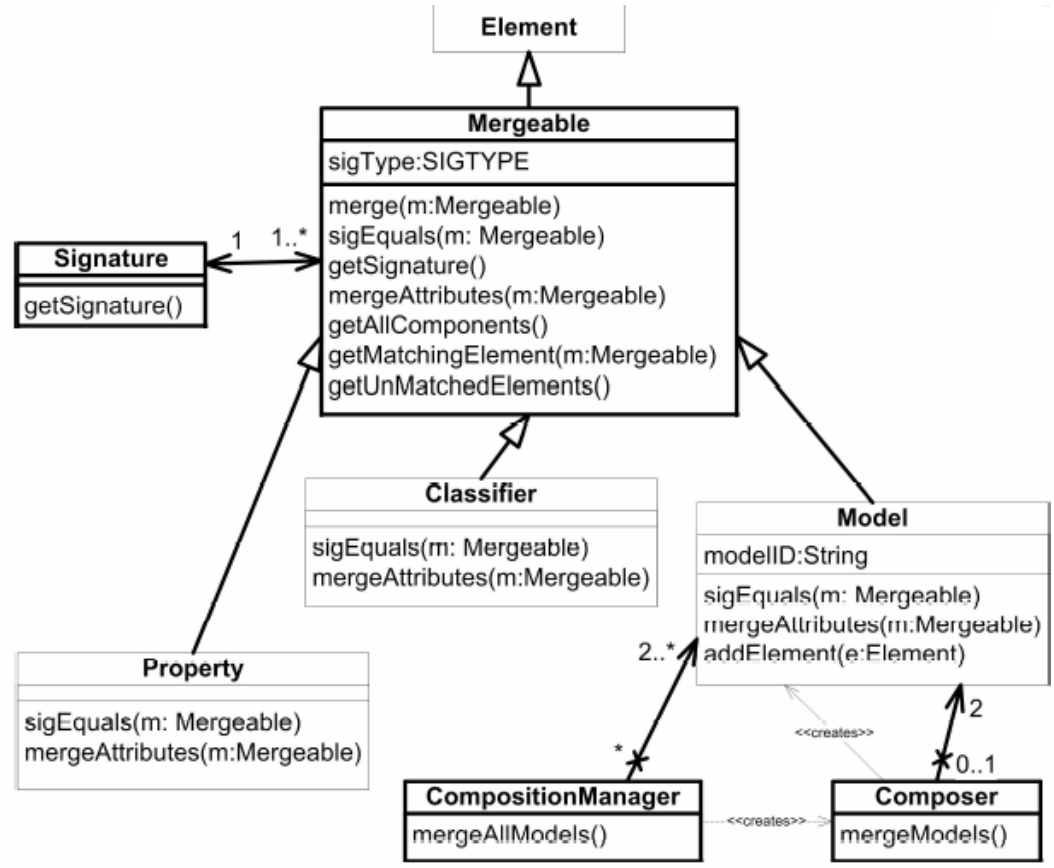


Figure 3.9 — Le métamodèle de l'approche AAM

précédemment. Quand le modèle composé est produit, l'opération `mergeAllModels` associe le modèle produit et un autre modèle à l'instance `Composer` et invoque à nouveau l'opération `mergeModels`. Ce processus se répète jusqu'à ce que tous les modèles soient composés [Anwar, 2009].

Les mécanismes de composition proposés par l'approche AAM constituent son point fort. Ces mécanismes sont implémentés avec le langage Kermeta [France et al., 2004], et mis en œuvre dans l'outil Kompose [France et al., 2007]. Toutefois, il est à signaler que le concepteur doit prendre en charge l'identification des problèmes de composition en spécifiant les directives de composition appropriées à appliquer.

3.5.4 La composition de modèles dans l'approche de Muller et al.

L'approche de composition proposée par Muller et al. [Muller, 2006], [Muller et al., 2007] repose sur la notion de *composant de modèle*. Elle repose sur la notion d'assem-

blage d'un ensemble de modèles paramétrés, articulés autour d'un *modèle de base*, pour concevoir un système. Elle propose une structuration du système par fonctionnalités représentées par des *composants de modèles*. Un *composant de modèle* est un modèle générique paramétré décrivant une fonctionnalité générique qui peut être réutilisée et adaptée selon un scénario d'application. La construction d'un système peut être réalisée par assemblage d'un ensemble de *composants de modèles* issus de différentes sources.

Cette approche propose deux modes d'assemblage : le mode de *fusion* et le mode de *structuration par Vues*.

- Le *mode fusion* fusionne le *modèle de base* avec des *composants de modèles* modélisant les différentes fonctionnalités pour obtenir un modèle à objet classique. L'identification des éléments, à fusionner, contient le nom de l'élément et l'élément qui les contient (par exemple : pour un attribut l'identificateur est défini par son nom et le nom de sa classe).
  Des conflits peuvent apparaître si des éléments de même nom doivent être ajoutés au même *modèle de base*. A cet effet cette approche peut adopter une stratégie de renommage des éléments à fusionner comme l'utilisation de directives de composition [Raghu et al., 2006] ou l'utilisation de mécanismes de priorité [Bouzitouna and Gervais, 2004]. Le principe de cette fusion est d'ajouter tous les éléments de la fonctionnalité décrite par le *composant de modèle* et qui font partie du modèle paramétré, au modèle auquel le composant est appliqué. Ainsi, les attributs et opérations définis par les composants sont ajoutés aux classes correspondantes, et il en est de même pour les classes et associations introduites par les composants. Le modèle obtenu avec ce mode d'assemblage est un modèle objet conventionnel qui représente le modèle de conception global du système. Cependant, cette représentation ne permet pas de conserver la structuration initiale du système selon ses différentes fonctionnalités.
- Le *mode structuration par Vues* permet d'obtenir un modèle selon une structuration par vues [Muller et al., 2003]. Les entités des différentes vues sont mises en correspondance avec les entités du *modèle de base*. Pour déduire la vue correspondant à un *composant de modèle*, on procède à la substitution des éléments paramètres par les éléments correspondants du *modèle de base* [Anwar, 2009].

La figure 3.10 illustre l'application du *composant de modèle* de gestion de ressources à un *modèle de base* décrivant un système de location de véhicules. Le but est de lui rajouter la fonctionnalité de gestion de stocks présenté dans la figure 3.11 par l'application de la relation stéréotypée *apply* qui permet de mettre en correspondance le modèle fourni (*modèle du composant*) et le modèle requis (*modèle de base*). Ainsi, les classes *Stock* et *Ressource* du composant sont respectivement mises en correspondance avec les classes *Agence* et *Vehicule* du *modèle de base*. De même, les attributs *identifiant* et *ref* sont mis en relation avec les attributs *nom* et *immatriculation* de la figure 3.10.

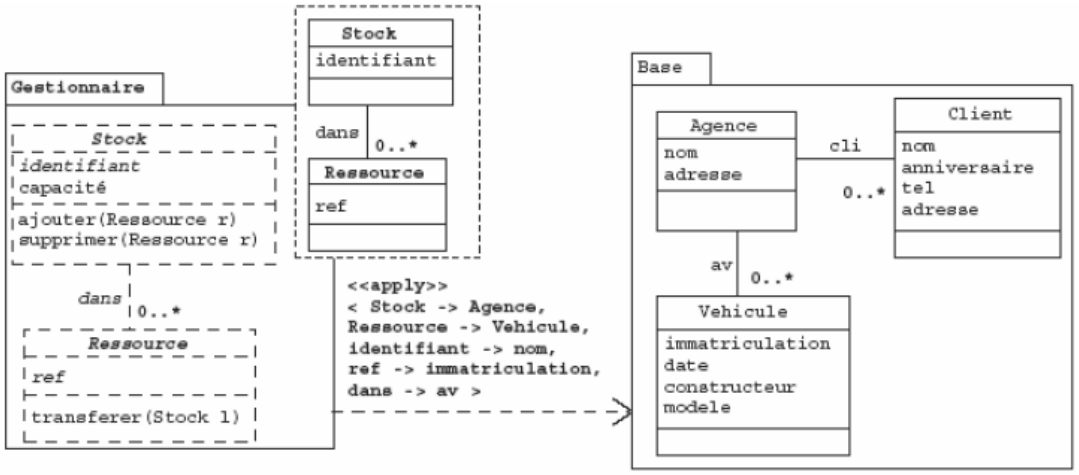


Figure 3.10 — Application de la gestion de ressources au système de location de véhicules [Muller, 2006]

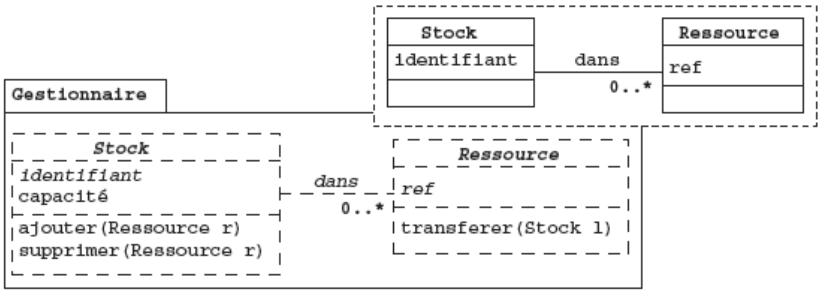


Figure 3.11 — Composant de gestion de ressources [Muller, 2006]

Signalons que dans cette approche le mécanisme de substitution des éléments n'est pas explicitement détaillé. De plus, le concepteur doit prendre en charge l'identification et la résolution des conflits qui peuvent apparaitre, car l'approche ne propose aucune stratégie de résolution de conflits.

3.5.5 La composition de modèles dans Atlas Model Weaver

L'ATLAS Modèle Weaver (AMW) est un framework générique de composition de modèles. AMW est élaboré par le groupe ATLAS, INRIA. Il permet d'établir des liens entre des éléments de modèles ou de métamodèles, et les associations entre les liens. Les liens sont stockés dans un modèle, appelé modèle de tissage. Ce modèle est conforme à un métamodèle. Les identifiants des éléments liés peuvent être enregistrés en utilisant

différentes méthodes d'identification. Le noyau de métamodèle de tissage d'AMW est illustré ci-dessous.

AMW est basé sur les principes de tissage et de transformation de modèles pour produire et implémenter l'opération de composition [Bézivin et al., 2006].

L'opération de tissage dans AMW comprend deux étapes.

1. La première étape consiste à spécifier dans un modèle de tissage les liens de composition entre les éléments des modèles d'entrée. Un lien de composition entre deux éléments indique une relation entre ces deux éléments, la sémantique de cette relation variant d'un domaine à un autre. Certains liens de composition représentent des opérations primitives de composition telles que la fusion, le remplacement ou la création. Il existe plusieurs façons d'implémenter ces primitives. Elles sont en général implémentées par des transformations écrites manuellement.
2. La deuxième étape génère automatiquement la transformation en appliquant une transformation d'ordre supérieur (HOT) [2] [DelFabro and Valduriez, 2007]. La transformation HOT se charge de transformer les liens de composition en patrons de code de composition spécifiques (cf. fig. 3.12).

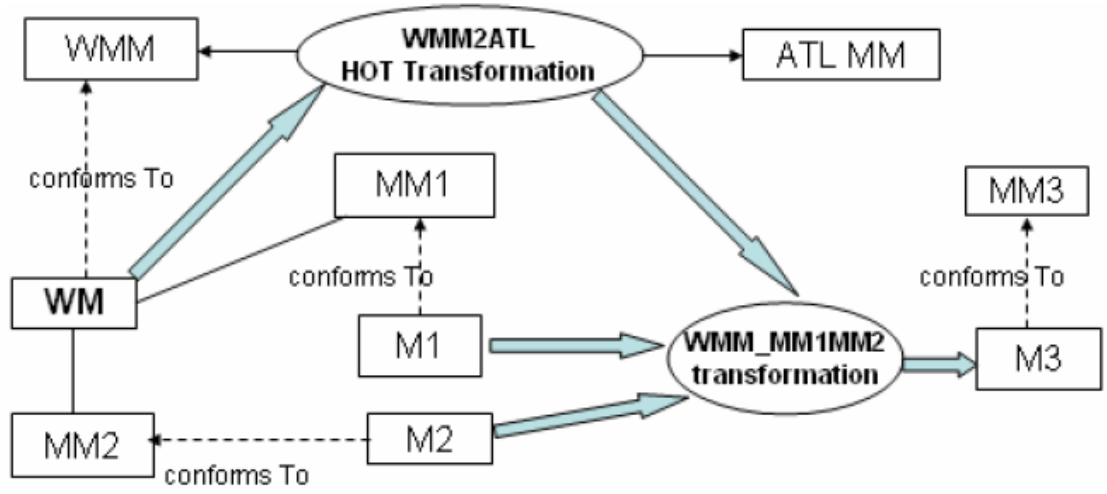


Figure 3.12 — Processus de génération de transformations dans AMW [DelFabro and Valduriez, 2007]

L'objectif principal du modèle de tissage est la représentation des différents types de liens entre les éléments des modèles à tisser. AMW prend en entrée deux modèles et produit en sortie le modèle composé. Le métamodèle de tissage de AMW est illustré

2. HOT : Higher Order Transformation. C'est à dire des transformations dont la source et/ou la cible sont elles-mêmes des transformations

dans la figure 3.13.
`WElement` est l'élément de base dont héritent tous les autres éléments. Il a un *nom* et

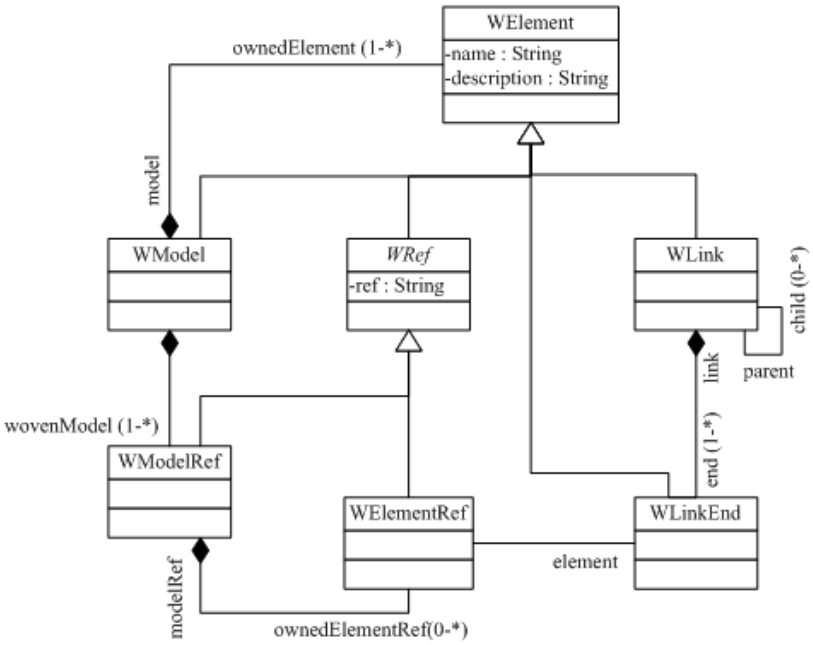


Figure 3.13 — Le métamodèle de tissage AMW [DelFabro and Valduriez, 2007]

une *description*. `WModel` représente l'élément racine qui contient tous les éléments du modèle. `WLink` désigne le type de lien. `WLink` a une référence `end` de `WLinkEnd` qui représente l'extrémité d'un lien. Chaque `WLinkEnd` référence un élément de `WElementRef`. L'attribut *ref* contient l'identifiant des éléments liés. `WElementRef` n'est pas référencé directement par `WLink` car il est possible de référencer le même élément de modèle par les points de terminaison (endpoints) de différentes relations (liaisons), par exemple, un élément de modèle peut participer à plus d'une expression de liaison (mapping expression). `WModelRef` est similaire à `WElementRef`, mais il contient des références à des modèles.

Dans AMW, la génération des liens de tissage dépend de plusieurs facteurs tels que le scénario d'application. Ces liens peuvent demander éventuellement l'interaction avec un utilisateur, soit la mise en œuvre de règles de correspondance heuristiques. De ce fait, la génération des modèles de tissage est une tâche qui n'est pas facilement automatisable.

Il y a un manque remarquable de consensus ou de standard qui définisse un métamodèle de tissage en mesure d'exprimer toutes les sémantiques des liens de composition. Cependant, le métamodèle de tissage de l'approche AMW, qui répond aux exigences communes pour la gestion des liens [DelFabro et al., 2005], est un métamodèle de base

qui supporte un mécanisme d'extension capable de l'adapter pour un scénario d'application donné.

Nous pouvons dire que l'approche AMW favorise la réutilisation grâce au mécanisme d'extension de son métamodèle générique de tissage. Un prototype de AMW a été implémenté sous forme d'un plug-in Eclipse et fait partie du projet GMT [DelFabro et al., 2006]. Dans cette optique, nous avons étendu le métamodèle de tissage de l'approche AMW, pour l'utiliser dans le contexte de notre travail. Cette extension est décrite en détail dans le chapitre 4.

3.6 Conclusion

Ce chapitre a permis de présenter une étude des approches représentatives du domaine de la composition et du tissage de modèles. Bien que la composition et le tissage de modèles ont été abordés dans de nombreux travaux de recherche, il n'y a pas encore de standard relatif à la composition et au tissage de modèles.

Les approches existantes de composition et de tissage de modèle ne satisfont pas entièrement nos besoins pour la composition de modèles dans AspeCiS, plus particulièrement dans sa deuxième phase. Notre proposition de tissage de modèle vise à produire les modèles des exigences coopératives. Pour cela le processus de tissage proposé étend celui proposé par Atlas Model Weaver (AMW). Il propose un profil UML pour modéliser les aspects qui devront être tissés avec les modèles existants. Ces aspects permettent une fois tissés, selon le modèle de tissage proposé, de réutiliser les modèles existants pour produire les modèles des exigences coopératives.

Le chapitre suivant présente en détail notre approche d'élicitation et de modélisation des exigences dans un système d'information coopératif.

Troisième partie

Une approche orientée Aspect d'élicitation et de modélisation des exigences dans les organisations multi-entreprises

4 Une approche orientée Aspect pour l'élicitation et la modélisation des exigences pour un système d'information coopératif

Ceux qui réussiront seront ceux qui arriveront à agir ensemble en pensant différemment (inconnu).

4.1 Introduction

Les entreprises de nos jours évoluent dans des environnements caractérisés par l'intensification de la concurrence, les changements des demandes clients et la performance. Dans ce contexte, et pour faire face à ces défis, la coopération inter-entreprises est une des solutions envisageables. Dans cette optique, si une organisation souhaite atteindre un nouvel objectif mais n'a pas toutes les compétences nécessaires et ne dispose pas des ressources (humaines, technologiques, financières) suffisantes pour réaliser seule cet objectif, elle décide de coopérer avec d'autres entreprises. Ces entreprises unissent leurs compétences et ressources, selon une forme de coopération, pour répondre aux opportunités. Ces formes d'organisation ont un impact important sur les systèmes d'information. En particulier, la flexibilité et l'ouverture vers l'environnement deviennent des enjeux majeurs dans la conception de ces systèmes d'information, ainsi que la réutilisation.

Nos travaux de recherche s'intéressent à la question des coopérations inter-entreprises en considérant que le système d'information est un élément central de cette problématique. On cherche à développer un système d'information qu'on qualifie de coopératif

pour supporter cette coopération. Mais, on envisage de ne considérer que des systèmes construits à partir de systèmes (pré-)existants.

L'une des solutions pour le développement de ce type de système d'information (SI), consiste à composer plusieurs SIs existants selon une stratégie de coopération pour répondre aux différentes tâches de la coopération. La composition des SIs est un domaine d'activité et de recherche. Le résultat de cette composition produit ce qu'on appelle un *Système d'Information Coopératif* (SIC).

Comme nous l'avons montré dans le premier chapitre, le paradigme aspect offre des possibilités importantes de réutilisation. C'est dans ce contexte que se situe le travail de cette thèse. De ce fait, nous avons constaté qu'il est bénéfique de proposer une approche, orientée aspect, de développement d'un nouveau SIC capable de supporter les taches complexes et ceci en réutilisant des artéfacts des SIs existants. A cet effet, ce chapitre est consacré à la présentation de cette approche dénommée AspeCiS (Aspect oriented approach to develop a Cooperative Information System).

L'approche AspeCiS [Amroune et al., 2011] propose une démarche d'élicitation et de modélisation des exigences du SIC à développer. Le développement d'un SIC commence par une phase d'élicitation des exigences, elle est suivie par une deuxième phase de modélisation des exigences élicitées.

Avant de présenter AspeCiS, nous commençons par rappeler dans la section 4.2 les concepts de base de la coopération inter-organisationnelle. Dans la section 4.3 nous présentons les principales motivations pour une nouvelle approche d'élicitation et de modélisation des exigences d'un SIC. Dans la section 4.4, nous détaillons les concepts de base sur lesquels repose AspeCiS. Nous décrivons dans la section 4.5, les différentes étapes ainsi que les processus d'AspeCiS, nous présentons le tissage de modèles dans AspeCiS dans la section 4.6 avant de conclure dans la section 4.7.

4.2 La coopération inter-organisationnelle : concepts de base

4.2.1 La coopération

Selon [Barnard, 1983], la coopération est le moyen de dépasser les limites de l'action individuelle. En ce sens, les organisations optant pour un mode de fonctionnement coopératif attendent en retour une minimisation des risques et une réduction de l'incertitude, ainsi qu'un accroissement de la performance industrielle [Campagne and Sénéchal, 2002].

L'encyclopédie *Universalis* définit la coopération comme *le fait, pour une personne, de s'adonner consciemment à une activité complémentaire de celles d'autres personnes dans le cadre d'une* ***finalité commune****, dans un groupe donné.* L'origine étymologique du terme coopérer est l'association de la racine ***opérer*** et du suffixe ***co*** qui donne la signification de *travailler ensemble.* Selon le dictionnaire Larousse coopérer est défini comme *agir conjointement.* C'est aussi le cas de la définition théorique commune qui considère la coopération comme *des liens que construisent entre eux des agents en vue de réaliser, volontairement, une œuvre commune* [Dejours, 1993].

Une définition adaptée à la coopération entre organisations est donnée par Boughzala [Boughzala and Ermine, 2004]. Pour ce faire, il part des trois traits extraits de la définition de la coopération en général, pour en extraire les conséquences et essayer de trouver les caractéristiques de la coopération inter-organisationnelle. Ces traits sont : le but commun, la forme consciente de la coopération et la résolution de problèmes.

Le premier trait : le but commun, pour qu'une coopération soit totale, il faut que le travail collectif corresponde à un but commun. La résolution de problèmes constitue également l'un des piliers de la coopération inter-organisationnelle. Résoudre un problème en groupe implique l'engagement de tous les membres du groupe dans cet objectif. Il faut qu'il y ait une volonté de coopérer, il faut que tous les partenaires soient conscients qu'ils ont à accomplir un objectif commun ou à résoudre des problèmes pour y parvenir. D'où le deuxième trait de la coopération : la forme consciente de la coopération, ou le degré d'engagement dans la coopération.

Ainsi, la coopération inter-organisationnelle peut donc être définie comme une situation où deux ou plusieurs agents partenaires, sous l'égide de contrats, mettent en commun des ressources et des moyens complémentaires pour la résolution de problèmes afin d'accomplir une ou plusieurs activités en commun. Ces agents communiquent entre eux afin de coordonner leurs tâches. [Hammami, 2003] a recensé quatre types de relation de coopération :

- les relations de coopération verticales : ce type de relation désigne un groupement dont les organisations agissent à des étapes successives de la chaîne de valeur (chacune des organisations joue le rôle de fournisseur et/ou client).
- les relations de coopération horizontales : elles sont bâties entre concurrents ayant décidé de collaborer ensemble pour atteindre un objectif commun. Elles peuvent concerner aussi bien des relations entre partenaires appartenant à des aires de marché différentes que des relations entre concurrents directs [Rullière and Torre, 1995]. Nous nous intéressons dans le cadre de cette thèse à cette forme de coopération..
- les relations de coopération diagonales : les partenaires de ces relations ne sont ni des concurrents ni des acteurs successifs de la chaîne de valeur.

– les relations de coopération entre l'université et l'industrie : elles mettent l'accent sur le fait que les relations de coopération entretenues par les entreprises dépassent le domaine industriel et commercial et s'étendent à l'ensemble de leur environnement [Tapon, 1989].

4.2.2 Pourquoi coopérer ?

Dans les relations inter-organisationnelles, un fonctionnement coopératif est motivé lorsque les organisations partagent des objectifs et des intérêts communs, et/ou des compétences et des points de vue, et/ou des ressources [Campagne and Sénéchal, 2002]. L'objectif de la coopération au sein d'actions collectives est alors de concevoir et produire mieux et plus vite des produits et services innovants. Les bénéfices de la coopération sont alors partagés par l'ensemble des acteurs.

Plusieurs raisons expliquent la coopération inter-organisationnelle, [Rullière and Torre, 1995] en distinguant sept :

1. la recherche d'économies d'échelle afin de profiter des synergies.
2. la modification et l'atténuation des règles de concurrence.
3. le retournement à son avantage du rapport de force concurrentiel en s'alliant avec des partenaires contre son principal concurrent.
4. la lutte contre l'incertitude et le partage des risques inhérents aux transactions marchandes ou à l'intégration.
5. le partage des coûts liés à des projets de développement importants entre les cocontractants, en vue d'accroître leur compétitivité cumulée.
6. la recherche d'un accès, au niveau international, à de nouveaux marchés, jusqu'alors fermés.
7. le transfert de savoir-faire et de technologies sans en abandonner les droits de propriété associés.

L'organisation doit donc identifier ses besoins et mettre en place la relation de coopération qui répond le mieux à ses attentes. Pour ce faire, elle doit veiller à suivre un processus organisé du début de la conception jusqu'à son terme, afin de garantir que la bonne stratégie ait été mise en place avec le bon partenaire, et que la coopération soit suffisamment bien soutenue pour atteindre les objectifs de départ [Lehoux et al., 2008].

4.3 Les motivations pour une nouvelle approche d'élicitation des exigences orientée aspect d'un système d'information coopératif

S'il est vrai que réutilisabilité, évolutivité et maintenance sont des propriétés des approches Objet et Aspect, il est envisagé, dans notre travail, de ne considérer que des systèmes construits à partir de systèmes (pré-)existants. De tels besoins sont rencontrés dans les plates-formes d'intégration de différents SIs d'entreprise, dans les systèmes basés sur l'intégration de données et dans les systèmes construits à partir de sous services (web services).

Une approche classique consisterait à analyser, concevoir puis développer le nouveau système par un ensemble de modules logiciels s'appuyant sur les sous systèmes composants. L'approche envisagée consiste à analyser les nouvelles exigences comme des aspects (précoces) ensuite tissés (propagés) sur les systèmes existants (plus précisément sur leur partie visible et accessible). Dans cette approche les exigences fonctionnelles du nouveau système sont appréhendées avec une technique qui habituellement ne concernent que les aspects non fonctionnels (persistence, parallèlisme, trace, etc).

AspeCiS est développé dans l'intention de :

1. ne pas mélanger dans le système produit les fonctionnalités pré-existantes des nouvelles fonctionnalités.
2. fournir un degré de *réutilisation fonctionnelle* qui pourrait permettre de *tisser* à nouveau ces mêmes fonctionnalités sur d'autres systèmes préexistants (réellement différents ou les mêmes dans une autre version etc..).
3. contribuer à promouvoir le paradigme Aspect surtout lors des phases amonts de développement logiciel.

Pour cela, AspeCiS tente de répondre à un ensemble de questions :

- quelle méthodologie d'analyse proposer pour capturer les nouvelles exigences sous forme d'aspects ?
- pourra-t-on proposer des types d'aspects fonctionnels ? y retrouvera-t-on les modèles classiques (ex : modèle, vue, contrôle) ?
- quels sont les points de jonction pour tisser des aspects fonctionnels sur un système existant ? doit-on envisager un ensemble prédéfini de points de jonctions ? un langage de définition de points de jonctions ?
- qans quelles conditions, un tissage fonctionnel est-il réapplicable ou réutilisable ?

4.4 Présentation des concepts de base d'AspeCiS

La réutilisation se définit comme une approche de développement de systèmes selon laquelle il est possible de construire un nouveau système, à partir des systèmes existants produits à l'occasion de développements antérieurs [Krueger, 1992]. Cette approche s'oppose aux approches usuelles de développement dans lesquelles la construction d'un nouveau système part de zéro et nécessite de tout reconstruire à chaque fois.

Pour être efficace, la réutilisation doit être considérée dès les phases amonts (IE, Analyse et de conception) de construction du système. Pour cela, le paradigme Aspect se présente comme une voie prometteuse. Nous tentons dans cette approche de construire un nouveau système en réutilisant des artéfacts des systèmes existants par une approche Aspect. C'est dans un contexte de réutilisation que se situe AspeCiS.

AspeCiS est structurée en trois grandes phases comme le montre la figure 4.1. La première est destinée à éliciter et analyser les exigences du système à développer en termes d'***Exigences Coopératives*** (ECs). La seconde phase élabore les modèles des ECs en se basant sur les modèles des exigences existantes. Quant à la troisième phase, elle sert à l'implémentation (cette phase n'est pas développée dans le contexte de cette thèse). Dans AspeCiS, nous avons défini trois types d'exigences, qui sont présentés en détail dans la section suivante.

4.4.1 Les catégories d'exigences exploitées dans AspeCiS

Quatre catégories d'exigences sont définies dans AspeCiS. Les ***Exigences Existantes (EEs)***, les ***Exigences Aspectuelles (EAs)***, les ***Exigences Supplémentaires (ESs)*** et les ***Exigences Coopératives (ECs)***. Avant de présenter en détail AspeCiS, nous commençons par donner des définitions à ces types d'exigences, en s'inspirant des définitions d'exigences qui existent dans la littérature [Nuseibeh and Easterbrook, 2000].

> ***Une Exigence Coopérative (EC)*** *est une exigence de but définie à partir des EEs et EAs, elle présente quelles parties des exigences des SIs existants seront réutilisées et composées, et quelles parties doivent être nouvellement développées (voir fig. 4.2).*

> ***Une Exigence Existante (EE)*** *est un énoncé de service ou une contrainte fourni par un système existant, qui défini la façon dont le système doit réagir à des entrées particulières et comment il doit se comporter dans une situation particulière.*

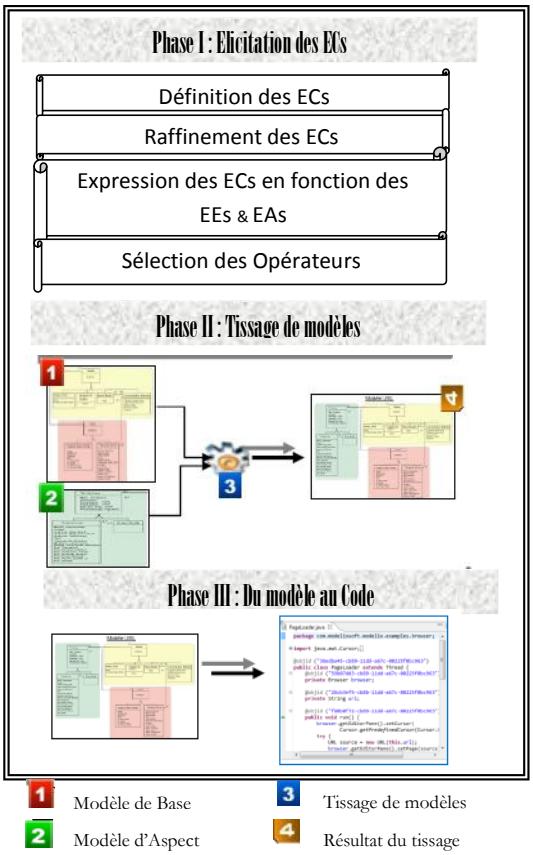


Figure 4.1 — Schéma général de l'approche AspeCiS

Exigence Aspectuelle (EA) *est un ensemble de comportements destinés à être tissés sur des EEs afin de les réutiliser. Elle décrit une partie des exigences à développer.*

Une Exigence Supplémentaire (ES) *est définie comme étant une exigence qui ne peut pas être obtenue par une réutilisation simple des EEs. Elle est le résultat de la combinaison des* ***EEs*** *avec des* ***EAs****.*

AspeCiS définit aussi deux nouveaux concepts qui sont les *Opérateurs de tissage* (OP-T), et les *Opérateurs de Composition* (OP-C) :

un Opérateur de Tissage (OP-T) *décrit la manière de tisser des EAs avec des EEs.*

un Opérateur de Composition (OP-C) *décrit la manière de composer par des conjonctions et/ou des disjonctions des exigences pour définir une*

nouvelle EC.

Il est important de noter que, à ce niveau d'abstraction, ces opérateurs sont considérés comme étant des exigences comme il est illustré dans la figure 4.2, qui représente un métamodèle de l'EC. Une explication de cette considération vient du fait que l'exigence *modifier une exigence* explique de manière générale le *pourquoi* des opérateurs de tissage, et qui servent à apporter les modifications appropriées des EEs afin de les réutiliser. A cet effet, et à ce niveau d'abstraction, les opérateurs de tissage sont considérés comme un type d'exigence. Le même raisonnement s'applique pour les opérateurs de composition, l'exigence *composer des exigences* explique aussi le *pourquoi* des opérateurs de composition, ce qui nous permet de les considérer aussi comme un type d'exigence.

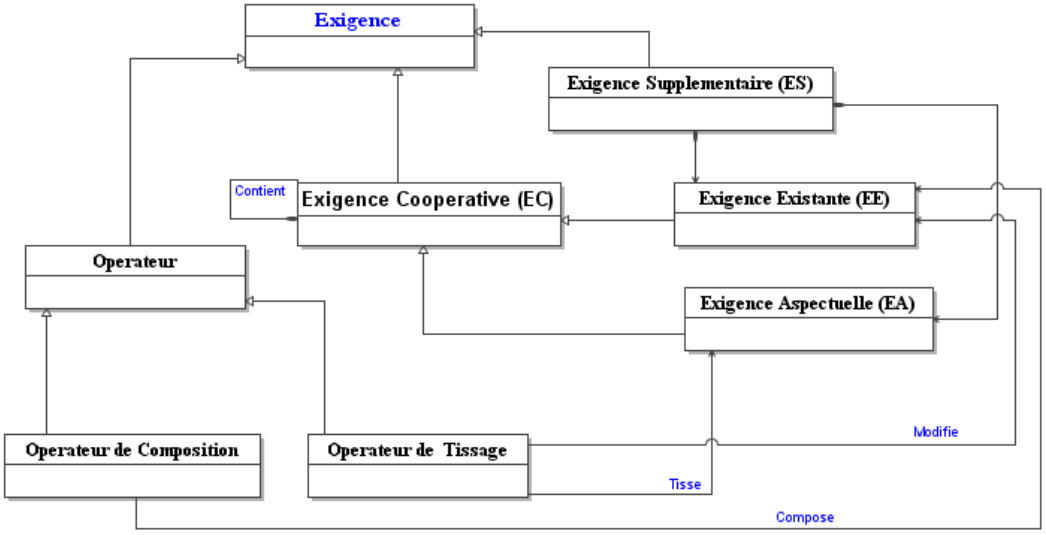


Figure 4.2 — Le métamodèle de l'Exigence Coopérative (EC)

Le métamodèle de l'EC (cf. fig. 4.2) montre bien que les quatre types d'exigences sont des types d'exigences. A noter que les EEs sont classées en deux catégorie :

1. catégorie 1 : les EEs qui sont utilisées telle quelles dans la définition de l'EC.
2. catégorie 2 : les EEs qui avec des modifications spécifiques peuvent être utilisées dans la définition de l'EC.

Les modifications nécessaires de ces EEs sont effectuées par les opérateurs de tissage *(Operateur de Tissage)*. Les *Opérateurs de Composition* ont pour rôle la composition des EEs. Cette composition consiste en la conjonction ou la disjonction de ses exigences. A ce niveau d'abstraction l'*Operateur de Tissage* et l'*Opérateur de Composition* sont considérés comme des exigences.

La modification des EEs est considérée par AspeCiS comme le ***tissage*** d'un nouveau comportement décrit par l'EA sur les EEs, ou d'altération des comportements dans les EEs.

4.5 Les différentes phases d'AspeCiS

AspeCiS couvre principalement les deux premières phases du développement de SIC qui sont l'analyse et la modélisation. L'objectif de la phase d'analyse est la définition d'un ensemble d'ECs exprimées en fonction d'un ensemble d'EEs combinées éventuellement avec des EAs. Les sections qui suivent détaillent cette phase.

4.5.1 L'élicitation et l'analyse des exigences dans AspeCiS

L'élicitation et l'analyse des ECs constitue la première phase d'AspeCiS. Elle est composée de quatre sous phases qui sont : la définition, le raffinement, la formulation des ECs en fonction des EEs et des EAs et la sélection d'opérateurs de tissage et de composition. Cette phase prévoit aussi la résolution des conflits qui peuvent apparaître lors de la composition des EAs.

4.5.1.1 La définition d'une EC

L'élicitation des exigences consiste en la collecte, la capture, et la découverte des exigences à partir d'une variété de sources. Dans la littérature plusieurs techniques d'élicitation des exigences sont définies. Le choix de la technique d'élicitation dépend du temps et des ressources disponibles au niveau des ingénieurs des exigences, ainsi que le type d'exigence à découvrir. L'équipe d'IE choisira la technique convenable selon les classifications des techniques d'IE présentées par B., Nuseibeh [Nuseibeh and Easterbrook, 2000].

- Techniques traditionnelles : parmi ces techniques, l'interview, l'introspection et l'analyse des tâches.
- Techniques de groupe : le brainstorming, la construction d'une histoire collective, la conception collaborative sont des exemples de cette catégorie de d'élicitation.
- Techniques de la psychologie cognitive : la psychologie cognitive est l'examen de la façon dont les gens comprennent, l'analyse des protocoles et le tri de cartes sont des exemples de ces techniques.
- Techniques contextuelles : elles sont vues par Nuseibeh & Easterbrook [Nuseibeh and Easterbrook, 2000] comme des alternatives aux techniques traditionnelles et aux techniques basées sur la psychologie cognitive. Comme leur nom l'indique, elles

mettent un accent particulier sur le contexte et l'environnement dans lesquels le système futur sera utilisé, c'est-à-dire, l'environnement du monde réel. Parmi ces techniques figurent l'ethnographie, les cas d'utilisation et le prototypage.

4.5.1.2 Le raffinement des ECs

Le processus de raffinement, proposé dans AspeCiS, permet d'éviter certains problèmes liés à la définition des ECs, tels que l'ambiguïté qui est due à *plusieurs interprétations possibles d'une même EC*, la redondance qui émerge lorsque *plusieurs EC expriment la même chose* et les incohérences qui sont dues à la *contradiction entre ECs*. Ce processus de raffinement est composé de deux actions qui sont la décomposition et l'inférence.

La décomposition des ECs. Cette action consiste à décomposer les ECs qualifiées d'exigences de haut niveau, en un arbre d'exigences élémentaires (non décomposables). Le processus de la décomposition (décrit ci-dessous) d'une EC donne un arbre d'EEs et EAs reliées par des conjonctions (ET) ou des disjonctions (OU). Cependant, il est important de noter que pour la définition des ECs, certaines EEs feuilles peuvent être utilisées sans aucune modification, alors que d'autres nécessitent, des modifications qui sont assurées par les opérateurs de tissage introduits précédemment.

Le processus de décomposition : le processus que propose AspeCiS, pour décomposer une EC, se fait en deux niveaux.

Niveau 1 : à ce niveau on décompose une **EC** en un arbre d'exigences existantes **EEs** et d'exigences supplémentaires (**ESs**) comme le montre la figure 4.3.

Niveau 2 : le deuxième niveau de la décomposition sert à développer les feuilles (**ESs**), issues de la décomposition de la première étape, en un ensemble d'EEs et d'EAs. A cet effet, les (**ESs**) sont le résultat du tissage d'EAs avec des EEs.

La relation d'inférence des ECs : AspeCiS utilise la relation d'inférence proposée par Mylopolus et al. dans [Jureta et al., 2010] et définie comme suit :
quand une exigence EC est la conséquence immédiate d'un autre ensemble d'exigences ECi, (i=1..n), on dira que EC est la conclusion et les EEi , (i=1..n) sont appelées les prémisses. Dans ce cas on dira que EC est **déduite** des exigences (ECi , i=1..n). La ***relation d'inférence des ECs*** permet :

1. d'éviter la redondance des ECs. L'utilisation de cette relation évite de définir des ECs pouvant être déduites des autres ECs.
2. d'éviter le problème d'ambiguïté.

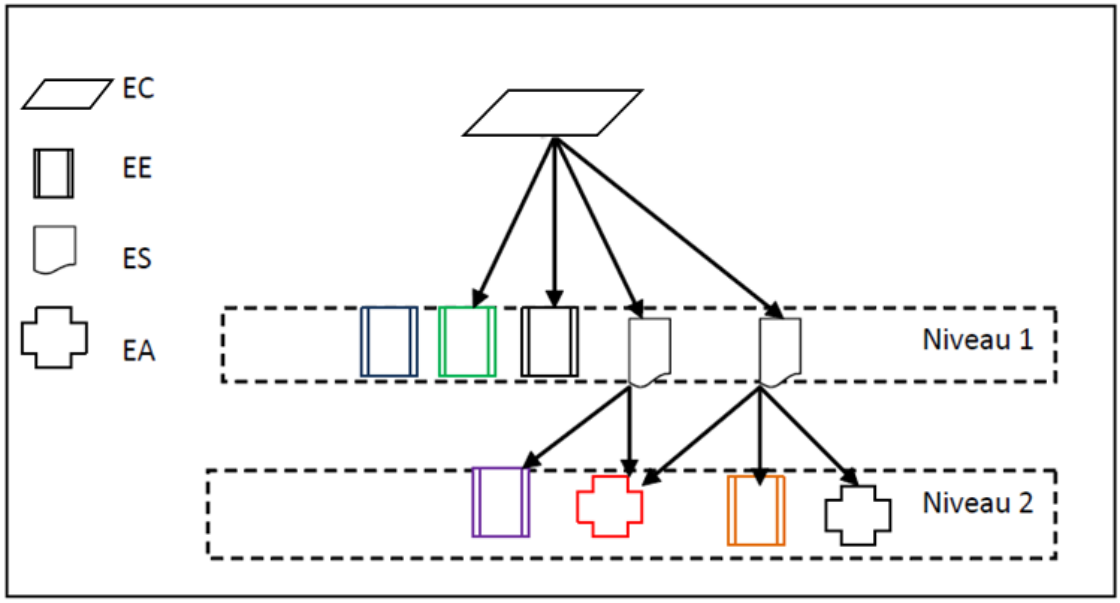


Figure 4.3 — L'arbre de décomposition d'une EC

3. de permettre un gain de coût et de temps de développement.

4.5.1.3 La formulation des ECs en fonction des EEs et des EAs

Une fois les ECs définies, cette sous-phase consiste à définir les EEs et les EAs intervenant dans la définition des ECs, c'est-à-dire exprime les ECs en fonction des EEs et des EAs. Le tableau 4.1 est un exemple tableau de bord qui donne pour chaque EC les EEs et les EAs impliquées dans sa définition. Chaque ligne de ce tableau donne l'ensemble des EEs et les EAs intervenant dans la définition d'une EC. Ce tableau est construit sur la base du résultat du processus de décomposition cité précédemment. A titre d'illustration, la première ligne du tableau 4.1 montre que l'exigence EC1, nécessite pour sa mise en œuvre la réutilisation par combinaison des exigences suivantes (EE1, EE2,EA1,EA2), avec EA1 et EA2 comme exigences aspectuelles.

A ce niveau, on doit montrer comment ***combiner*** les exigences EEi et les exigences EAs, pour répondre à une EC avec les modifications appropriées de certaines EEs ? La réponse est l'objet de la section suivante.

4.5.1.4 La sélection des opérateurs

Rappelons que les SIs existants sont sollicités dans la construction du nouveau SIC. Les exigences de chaque SI doivent alors contribuer à la définition des exigences du sys-

EEs/EAs / ECs	EE_1	EE_2	EE_3		....	EE_n	EA_1	EA_2	EA_3	.....	EA_k
EC_1	x						x				
EC_2		x				x		x			
EC_i	x							x			
EC_m	x	x				x					

Tableau 4.1 — Exemple de tableau de bord des ECs

tème à construire. A cet effet certaines EEs doivent être modifiées pour mettre en œuvre les ECs émergentes. Ces opérations de modification touchant les EEs, sont réalisées par les opérateurs proposés dans cette approche. Toutefois, il est important de signaler qu'une EA peut être tissée avec plusieurs EEs. De ce fait, les EAs sont de nature ***transversales*** par rapport aux EEs. On peut dire que :

1. le problème de réutilisation des EEs, consiste à bien définir l'opération de ***modification*** **des EEs**.

2. la modification des EEs est vue par notre approche comme le ***tissage*** des nouveaux comportements sur les EEs.

3. la solution consiste à définir un ensemble d'opérateurs que nous proposons dans AspeCiS, et qui vont agir sur les EEs.

Deux catégories d'opérateurs sont à prévoir, la première catégorie, est celle des ***Opérateurs de tissage (Op_T)***. Ces opérateurs tissent un ensemble de comportement, décrit par les EAs, sur certaines EEs. La deuxième est celle des ***Opérateurs de composition (Op_C)***, qui peuvent composer un ensemble d'exigences.

Le tableau 4.2, mentionne pour chaque ***Op_T*** quelles sont les EEi et les EAi sur lesquelles il agit.

Formellement, l'exemple suivant montre comment on peut appliquer les opérateurs de tissage et les opérateurs de composition pour définir une EC. Dans cet exemple quatre

EEs & EAs / OP_T	EE1	EE2	EE3	EE4	...	EEn	EA1	EA2	EA3	....	EAk
OP_T1	x	x					x	x			
OP_T2			x	x					x		
OP_Ti		x			x			x			
OP_Tm			x		x		x				x

Tableau 4.2 — Tableau de bord des Opérateurs

exigences (EEs) et trois exigences (EA) sont utilisées.

$$EC1 = Op\_T1(EE1, EA1) \tag{4.1}$$
$$EC2 = Op\_T1(EE2, EA2) \tag{4.2}$$
$$EC3 = Op\_C1(EC1, EC2) \tag{4.3}$$
$$EC4 = Op\_T2(EE3, EA3) \tag{4.4}$$
$$EC5 = Op\_T2(EE4, EA3) \tag{4.5}$$
$$EC6 = OP\_C2(EC5, EC4) \tag{4.6}$$
$$EC = OP\_C3(EC3, EC6) \tag{4.7}$$

Pour illustrer et consolider la description de la définition d'une EC, un scénario exprimé par un diagramme d'activité UML est donné dans la figure 4.4. Le diagramme d'activité de la figure est donc particulièrement adapté à la modélisation du cheminement de flots de contrôle et de flots de données. Il permet ainsi de représenter graphiquement le comportement d'une méthode ou le déroulement d'un cas d'utilisation. Le diagramme de la figure montre comment définir l'EC1 en montrant les opérations parallèles et en séries.

Un exemple de définition d'une EC en utilisant les opérateurs de tissage et de composition : pour donner des exemples sur les opérateurs proposés, nous supposons que nous voulons construire un nouveau SIC pour la gestion d'un projet de coopération, faisant intervenir plusieurs universités, afin d'assurer une formation doctorale. Chaque université est supportée par son SI existant. Le nouveau SI est construit sur la base des SIs existants de chaque université comme le montre la figure 4.5.

L'opérateur de tissage (Op_T) : ajouter une propriété à une EE, modifier une contrainte définie dans une exigence, sont des exemples d'EAs qui vont être tissées par des opérateurs de tissage sur certaines EEs. Dans le cas de trois SIs de gestion de

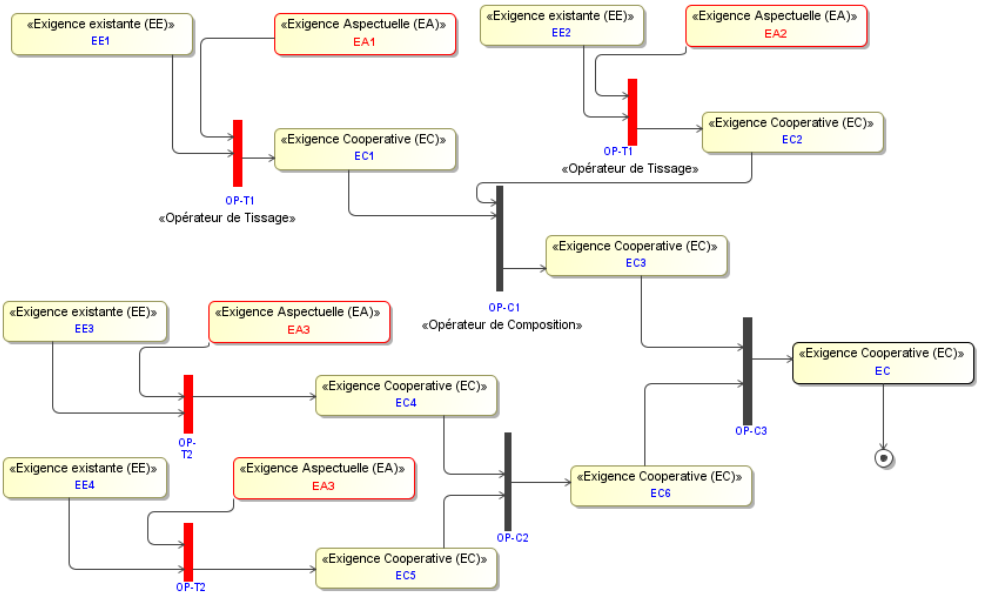


Figure 4.4 — Le diagramme d'activité de la définition d'une EC

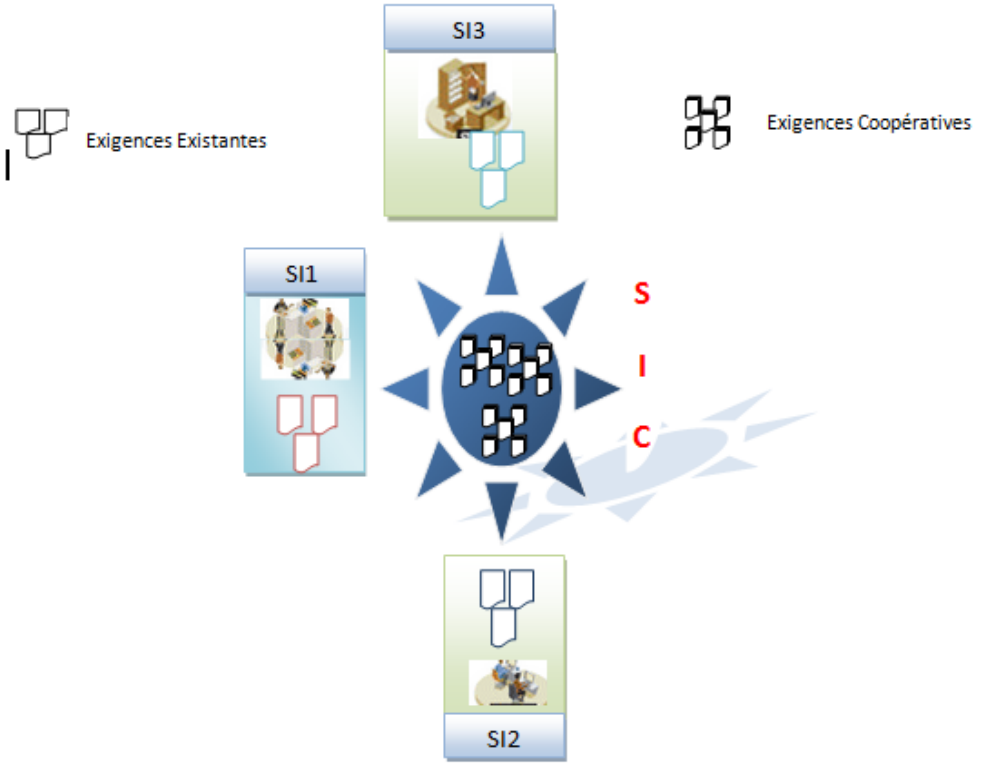


Figure 4.5 — Coopération de SIs pour développer un SIC

scolarité d'établissements universitaires : si les EE_i sont les exigences existantes définies au niveau des SI_i, supposons les cas suivants :

1. ***ajouter des propriétés à des EEs***.
   Les EE_1, EE_2, EE_3 définissent les deuxièmes inscriptions dans SI_1, SI_2, SI_3 respectivement.
   EE_1, EE_2, EE_3 = *Chaque étudiant peut s'inscrire dans une deuxième spécialité dans le même établissement.*
   L'EC_1 se définit comme suit :
   EC_1=*Chaque étudiant peut s'inscrire dans une deuxième spécialité dans le même établissement* ***à condition que*** *le volume horaire de la deuxième spécialité ne dépasse pas 50% du volume horaire de la première.*
   EC_1 = Combinaison(EE1, EE2, EE3), *avec l'ajout d'une propriété au niveau de chaque EE_i.*

   La définition de l'**EC_1** fait émerger une nouvelle exigence définie comme suit :
   EA1= *Comparer le volume horaire de deux spécialités.*
   On applique l'opérateur de tissage **Op_T1** sur les **(EE_1, EE_2, EE_3)** pour prendre en considération cette nouvelle propriété ***"à condition que"***, imposée lors de la définition de l'exigence coopérative EC_1 (*condition d'une deuxième inscription des étudiants*).

   Pour être conforme à la réglementation des doubles inscriptions, chaque SIs doit respecter ce règlement. Pour cela, ***l'Op_T1 tisse une nouvelle propriété définie par EA1 sur les EE_i existantes.***
2. ***modification de contraintes :*** l'EC_2 définit l'admission des étudiants comme suit :
   EC_2=*Chaque étudiant admis doit avoir 50% des crédits dont plus de 25% des crédits des unités fondamentales pour tous les établissements.*

   Pour l'admission des étudiants, les Exigences disponibles dans les systèmes existants sont les suivantes :
   EE_4 =*Chaque étudiant admis doit avoir 50% des crédits.*
   EE_5 = *Chaque étudiant admis doit avoir 50% des crédits dont 25% des crédits des unités fondamentales.*
   EE_6 = *Chaque étudiant admis doit avoir 50% des crédits.*

   On remarque que l'EE_5 définie dans le SI_2 est conforme à l'exigence coopérative (EC_2). Par contre les deux autres exigences EE_4 et EE_6 doivent prendre en considération la contrainte de 25% de crédit d'unité fondamentale. Le processus de décom-

position de l'EC_2 fait émerger une nouvelle exigence qu'on qualifie d'EA qui doit réaliser la contrainte des 25% des crédits fondamentaux.

A cet effet, l'EA2 = *vérifier la contrainte des 25% des crédits obtenus*, est introduite et sera prise en compte au niveau de tous les SIs existants.

Donc, l'opérateur de tissage ***Op_T2*** est appliqué sur les exigences (EE_4, EE_6) pour prendre en considération cette nouvelle contrainte lors de la définition de l'exigence EC_2.

L'EC_2 est définie comme suit :

$$EC2.1 = Op\_T2(EE4, EA2) \tag{4.8}$$

$$EC2.2 = Op\_T2(EE6, EA2) \tag{4.9}$$

$$EC2 = Op\_C1(EC2.1, EC2.2, EE5) \tag{4.10}$$

Op_C1 représente la disjonction (OU) introduite par le fait que les étudiants admis sont les étudiants admis des différents établissement (ie : des différents SIs).

On constate que l'Op_T2 tisse une nouvelle propriété sur les EE_i existantes (EE_4 et EE_6).

L'opérateur de composition (Op_C) : soient EE7, EE8, EE9 les exigences d'authentifications au niveau des SIs existants qui authentifient à leur manière leurs usagers (ex : nom, mail, pseudo), et l'exigence d'authentification du système coopératif (EC) qui pourrait accepter pour authentification :

- l'une des authentifications existantes, dans ce cas :
  EC= **OU**(EE_7,EE_8,EE_9)
- ou toutes les authentifications existantes dans ce cas EC= **ET**(EE_7,EE_8,EE_9)

*On considère que les opérateurs de conjonction **ET** et de disjonction **OU** sont des cas d'opérateurs de composition des exigences.*

Par exemple :

Si EE_7 est *l'authentification par login dans SI_1* et EE_8 est *l'authentification par mail dans SI_2*. L'exigence d'authentification du SIC pourrait-être définie par :

EC= *l'authentification se fait par login avec seulement deux tentatives, puis est suivie d'une authentification par mail.*

Le processus de décomposition de l'EC donne les exigences (EE_7, EE_8, EA). L'exigence EA est définie comme suit :

EA = *autoriser seulement deux tentatives d'authentification par login.*

En utilisant des opérateurs de tissage, l'exigence d'authentification (EC) est définie comme suit :

EC = **Op-C1(Op_T3(EE$_7$, EA) , EE$_8$)**.

Premièrement l'opérateur Op_T3 tisse l'exigence EA aspectuelle avec l'exigence existante EE_7 pour modifier l'exigence d'authentification du premier système SI_1, et ceci pour se conformer aux règles d'authentification définies par l'exigence EC (ie : modification du nombre de tentatives dans EE_7). Puis l'opérateur de composition Op-C1 (**ET**) compose l'authentification modifiée du premier système (ie : le résultat de Op_T3(EE_7, EA)) avec l'exigence EE_8 celle du deuxième système.

4.5.2 La résolution de conflit dans AspeCiS

Les ECs sont le résultat de la composition d'EEs et/ou d'EAs à l'aide d'opérateurs. Cependant, les EAs peuvent avoir des influences mutuelles négatives. A titre d'exemple si la définition d'une EC fait appel à une EA assurant la *sécurité* et une autre EA assurant la *rapidité* (le temps de réponse), il est clair que la première influe négativement sur la deuxième. Par voie de conséquence, un processus de résolution de conflits s'avère nécessaire. AspeCiS prévoit ce processus lors de la première phase afin de détecter et résoudre, le plus tôt possible, les différentes situations conflictuelles. Il est reconnu que l'utilisation de ce processus lors des phases amonts a plusieurs avantages tels que le gain en coût et en temps de développement. Cette section présente en détail le processus que propose AspeCiS pour la résolution de conflits inter exigences aspectuelles.

Le processus proposé [Amroune et al., 2012], cherche à donner un ordre de priorité aux EAs impliquées dans la définition des ECs, en utilisant une fonction mathématique dont l'un des paramètres est la priorité des parties ayant exprimées ces EAs (voir section 4.5.2). Dans le cas où cette fonction ne règle pas le conflit, d'autres critères sont utilisés.

La prioritisation des parties prenantes : La priorité où le poids des parties prenantes est un facteur important dans la résolution de conflit. Une question importante est comment peut-on donner des priorités aux parties prenantes. Afin de les prioritiser, nous avons utilisé le modèle développé par Michel et al. [Mitchell et al., 1997]. Ce modèle définit trois facteurs qui déterminent la priorité d'une partie prenante, et qui sont : *le pouvoir*, *la légitimité* et *l'urgence*.

Le *pouvoir* : une partie prenante possède un pouvoir si elle est en mesure de conduire un autre acteur à faire quelque chose qu'elle n'aurait pas réalisée autrement. Cependant, les parties prenantes peuvent également acquérir du pouvoir à travers les relations entretenues avec les autres acteurs du réseau social et institutionnel dans lequel elles évoluent. En effet, pour influencer les comportements de l'entreprise, les parties prenantes peuvent

utiliser directement leurs propres ressources comme elles peuvent utiliser les ressources de leurs alliés.

La *légitimité :* selon Mitchell et al. [Mitchell et al., 1997], la légitimité est définie comme une perception ou une supposition généralisée capable de rendre les actions d'une entité désirables, convenables et appropriées et de correspondre au système socialement construit de normes, de valeurs et de croyances.

L'entreprise prend en considération les demandes des parties prenantes qui ont un écho au niveau sociétal. Cette considération englobe les relations des parties prenantes avec l'entreprise et prend aussi en considération l'environnement institutionnel qui les entoure.

L'*état d'urgence des demandes :* pour catégoriser les parties prenantes, Mitchell et al., identifient un dernier critère, celui de l'état d'urgence des demandes. Il mesure le degré de la prise en compte immédiate par l'entreprise des exigences de ces parties prenantes. Pour mesurer le degré d'urgence, deux attributs sont définis à savoir : la contrainte au temps (« time sensitive ») exercée par les parties prenantes et la gravité, aux yeux des parties prenantes, de leur revendication ou de leur relation avec l'entreprise.

Les trois attributs (pouvoir, légitimité et urgence) sont attribués par les managers de l'entreprise à chaque partie prenante. Ainsi, l'attribution par les managers des attributs pour chaque partie prenante se base sur l'intention qu'ont les parties prenantes vis à vis du devenir de leurs relations avec l'entreprise ainsi que des actions qu'elles mènent avec ou contre les objectifs de cette dernière. Les pressions institutionnelles du champ dans lequel l'entreprise évolue influe considérablement sur la perception des managers de ces attributs aux parties prenantes. Selon le nombre d'attributs possédés par la partie prenante (pouvoir, légitimité ou urgence), les organisations se distinguent respectivement comme des parties prenantes latentes (latent stakeholder), des parties prenantes en attente (expectant stakeholder) et des parties prenantes définitives (definitive stakeholder). Cependant, une organisation doit avoir au moins un attribut pour se considérer par l'entreprise comme une partie prenante [Etzioni, 1964]. Chacune de ces catégories est ensuite décomposée en fonction de la combinaison des attributs (figure 4.6).

La partie prenante dormante (dormant stakeholder), qui fait partie des parties prenantes latentes, bien qu'elle possède le pouvoir afin d'imposer sa volonté à l'entreprise, son pouvoir demeure inutilisé. Puisque elle n'a que peu ou pas d'interaction avec l'entreprise à cause de l'absence de légitimité et demande urgente.

La partie prenante discrétionnaire (discretionary stakeholder) est considérée par le manager comme légitime mais ne possède pas le pouvoir d'influence sur l'entreprise et des demandes urgentes. Toute seule, elle ne peut engager de pression sur l'entreprise. La partie prenante revendicatrice (demanding stakeholder) ne possède pas le pouvoir et la légitimité, elle détient simplement des demandes urgentes à faire passer. Elle n'est

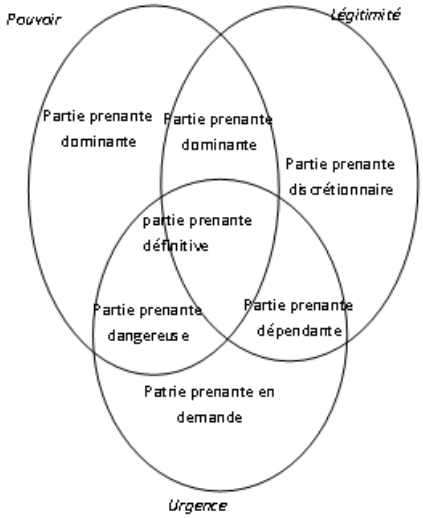


Figure 4.6 — Typologie des parties prenantes [Mitchell et al., 1997]

en mesure d'influencer les managers si elle agit seule. En utilisant la priorité des parties prenantes, le processus de résolution de conflit est structuré en quatre étapes. Les sections suivantes le présentent avec plus de détail.

Les différentes étapes du processus de résolution de conflit : ce processus est composé principalement de quatre étapes. La première détermine pour chaque EC les parties prenantes impliquées. Dans la seconde, le processus détermine la priorité assignée par chaque partie prenante aux EAs intervenant dans la définition de chaque EC. Quant à la quatrième étape, elle consiste à appliquer les critères définis pour la résolution de conflit. Ce processus est décrit comme suit.

Etape 1 : dans cette étape, une matrice est construite, indicée en ligne par les parties prenantes et en colonne par les ECs (M-PECs par abréviation). Elle indique les parties prenantes impliquées dans chaque EC, comme le montre la tableau 4.5.2. Si une cellule est marquée par le signe (x), cela veut dire que la partie prenante correspondante est impliquée dans l'ECi.

Etape 2 : pour chaque EC, on construit la matrice (Parties prenantes x EAs) appelée (M-PEA par abréviation). Cette matrice exprime dans sa deuxième colonne les priorités des parties prenantes selon le modèle de Michel et al., et dans les autres colonnes la priorité assignée par chaque partie prenante aux EAs exprimées (dans le tableau 4.4, $PP(EA_{i,j})$ désigne la valeur de priorité de l'EA(j) attribuée par la partie prenante (i)).

Partie prenante & ECs	EC1	EC2	...		ECn
Partie prenante$_1$	x	x	...		x
Partie prenante$_2$		x	...		
			...		
			...		
Partie prenante$_n$		x	...		

Tableau 4.3 — La matrice (Parties prenantes x ECs)

EAs & Partie prenante	Priorité PP	EA$_1$	EA$_2$			EA$_m$
Partie prenante$_1$	PP$_1$	PP(EA$_{1,1}$)	PP(EA$_{1,2}$)			PP(EA$_{1,n}$)
Partie prenante$_2$	PP$_2$	PP(EA$_{2,1}$)	...			
			...			
Partie prenante$_i$	PP$_i$		...			PP(EA$_{i,n}$)
			...			
Partie prenante$_n$	PP$_n$		...			PP(EA$_{i,m}$)
PP(EA$_i$)		PP(EA$_1$)				PP(EA$_m$)
Som-PP(EA$_i$)		Som-PP(EA$_1$)				Som-PP(EA$_m$)
NbrePP(EA$_i$)		NbrePP(EA$_1$)				NbrePP(EA$_m$)

Tableau 4.4 — La matrice (Parties prenantes x EAs)

Les trois dernières lignes de cette matrice marquées par **PP(EA$_i$)**, **Som-PP(EA$_i$)** et **NbrePP(EA$_i$)** dénotent :

1. **PP(EA$_i$)** est une fonction qui exprime comment attribuer la valeur de la priorité des EAs en fonction des poids des parties prenantes. Notons que :
   - PP_i est la priorité de la i-ème partie prenante.
   - EA$_i$ dénote la i-ème exigences aspectuelle ;
   - $PP(EA_{i,j})$ dénote le poids attribué par la i-ème partie prenante à l'exigence EA$_j$, $PP(EA_{i,j})$ appartient à (1 :très faible ; 2 :faible ; 3 :moyenne ; 4 :élevé ; 5 :très élevé) ;
   - PP(EA$_j$) dénote la priorité de l'EA$_j$ résultat de la fonction **PA** qui prend ses valeurs dans l'intervalle [1,5] ;
   - i=1..n, ou n, est le nombre de parties prenantes dans le système.

1. La fonction **PA**$_j$ est définie comme suit :

$$PA(EA_j) = \frac{\sum_{i=1}^{n} PP_i * PP(EA_{i,j})}{\sum_{i=1}^{n} PP_i} \tag{4.11}$$

EA & EA	EA_1	EA_2	...		EA_n
EA_1		$\ominus$	...		$\oplus$
EA_2	$\ominus$		...	$\ominus$	
			$\ominus$		
			...		
EA_n		$\oplus$	...	$\ominus$	

Tableau 4.5 — La matrice d'influence MI(EA x EA)

EA & EA	EA_1	EA_2	...		EA_n
EA_1		$\ominus$	...		
EA_2	$\ominus$	$\ominus$	...	$\ominus$	
			$\ominus$		
			...		
EA_n			...	$\ominus$	

Tableau 4.6 — La matrice des conflits MC(EA x EA)

2. La somme des poids des parties prenantes pour l'exigence EA_i est donnée par :

$$Som - PP(EA_i) = \sum_{j=1}^{n} PP_j \tag{4.12}$$

3. **NbrePP(EA_i)** dénote le nombre des parties prenantes ayant exprimés l'exigence EA_i.

Etape 3 : lors de cette étape, les EAs sont comparées deux à deux dans une matrice carrée symétrique appelée matrice d'influence MI(EA x EA). Les valeurs de cette matrice appartiennent à l'ensemble ($\oplus$, $\ominus$). Elle détermine le type d'influence entre les EAs. A titre d'exemple, si l'EA_i influe négativement sur l'EA_j, la cellule correspondante de la matrice d'influence est marquée par le signe ($\ominus$) et elle est marquée par le signe ($\oplus$) dans le cas contraire. La matrice d'influence est représentée dans le tableau 4.5. Cette étape est inspirée du travail de [Awais et al., 2003].

Etape 3.1 : l'algorithme de la résolution de conflit, présenté dans l'étape suivante, porte uniquement sur la matrice de conflit (MC) déduite de la matrice d'influence (MI). Elle contient les cas conflictuels marqués par le signe $\ominus$ dans la matrice MI (figure 4.6).

Etape 4 : la résolution de conflit : L'algorithme de résolution de conflits (fig. 4.7) a en entrée la matrice des conflits MC et en sortie la matrice de résolution de conflits (MRC). La matrice MRC est de même taille que la matrice MC. Les éléments de la matrice MRC appartiennent à l'ensemble { $\ll$, $\gg$ }.
Si MRC[i,j] = $\gg$, cela signifie que l'exigence EA_i et prioritaire par rapport à l'exigence EA_j. Par contre Si MRC[i,j] = $\ll$, cela signifie que l'exigence EA_j est plus prioritaire que l'exigence EA_i.

Cet algorithme utilise quatre critères de résolution de conflits. Si tous ces critères échouent, alors le recours au consensus entre les parties prenantes est nécessaire. Ces critères sont :

critère 1 : l'EA dominante est celle qui a la priorité la plus élevée, calculée par la fonction PP.
critère 2 : la somme des poids des parties prenantes détermine l'EA dominante dans ce cas.
critère 3 : le nombre des parties prenantes ayant exprimées un aspect détermine sa dominance.
critère 4 : la fréquence d'une EA est un facteur déterminant l'exigence dominante.

Etape 4.1 : la fréquence d'exigence aspectuelle : l'algorithme de résolution de conflit fait recours à la fréquence des EAs, au cas où les trois premiers critères échouent. Nous définissons la fréquence d'une EA comme suit :

> ***la fréquence d'une EA Freq(EA_i) :*** *est le rapport, entre le nombre de fois où cette exigence est impliquée pour la définition des ECs du système, sur le nombre total de cas conflictuels auxquels elle a participé.*

Une matrice de fréquences est établie à cet égard. La fréquence d'une EA est donnée par la formule suivante :

$$\boldsymbol{Freq(EA_i)} = \frac{Nbre(EA_i)}{NbreConf(EA_i)} \tag{4.13}$$

```
Input
MC: Matrice /* matrice de conflit de l'étape 3.1.*/
Output
MRC : Matrice /* la matrice de résolution de conflit.*/
Debut
Pour i := 1 to N faire
Pour J :=1 to (i-1) faire
Debut
Si MC[I,J] = "-" Alors
/*("utilisation du premier critère")*/
                    Si PP(EAi) > PP(EAj) Alors MRC[I,J]:= ">>"
                                                Sinon
                                               Si PP(EAi) < PP(EAj) Alors MRC[I,J]:= "<<"
                                                                        Sinon
/* ("utilisation du deuxième critère")*/
                                               Si Som-PP(EAi) > Som-PP(EAj) Alors MRC[I,J]:= ">>"
                                                                                    Sinon
                                               Si Som-PP(EAi) < Som-PP(EAj) Alors MRC[I,J]:= "<<"
                                                                                    Sinon
/* (utilisation du troisième critère")*/
                                               Si NbrePP(EAi) > NbrePP(EAj) Alors MRC[I,J]:= ">>"
                                                                                    Sinon
                                               Si NbrePP(EAi) < NbrePP(EAj) Alors MRC[I,J]:= "<<"
                                                                                    Sinon
/* "4 eme critère (utilisation du facteur de fréquence)" */
                                                 Si freq(EAi) > freq(EAj) Alors MCR[I,J]:= ">>"
                                                                                  Sinon
                                                 Si freq (EAi) < freq(EAj) Alors MCR[I,J]:= "<<"
/* (Trade Off)*/
                                                                    Sinon MRC:= "-"
  FinSi;


End.
```

Figure 4.7 — Algorithme de résolution de conflits

Nous pouvons dire, que cet algorithme retarde au maximum le recours au consensus pour la résolution des conflits, et ceci du fait que le consensus n'est pas toujours évident parce que chaque partie prenante veut voir ses points de vue retenus. Nous avons également proposé deux autres techniques de résolution de conflits [Amroune et al., 2014b], [Amroune et al., 2014a].

Après avoir défini les ECs y compris la résolution des cas conflictuels inter EA, la deuxième phase d'AspeCiS s'intéresse à la modélisation, laquelle sera présentée dans la section qui suit.

4.6 Le tissage de modèles dans AspeCiS

Le tissage de modèles dans AspeCiS constitue sa deuxième phase. A ce niveau, nous cherchons à établir les modèles des ECs, en se basant sur les modèles existants des EEs et les modèles des EAs. Cette phase a comme entrée un ensemble de modèles de base (modélisant les EEs) et un ensemble de modèles d'Aspects (modélisant les EAs). Elle donne comme résultat un ensemble de modèles d'ECs obtenus par un processus de tissage que nous définissons dans la section suivante. Nous supposons que tous les modèles issus de tous les SIs sont conformes au même métamodèle. Donc, l'hétérogénéité des langages de modélisation n'est pas considérée dans cette approche.

Le concept de tissage est utilisé pour supporter la composition des modèles. Ce concept n'est pas nouveau, plusieurs approches le défissent au niveau modèle. Cependant, plusieurs processus de tissage ont été proposés [Jayaraman et al., 2007], [Raghu et al., 2006], [Raghu et al., 2006], [Gorrieri and Wehrheim, 2006]. Dans le troisième chapitre nous avons présenté des approches utilisant la composition de modèles. Les approches citées sont les approches de Clarke [Siobhán, 2002], [Baniassad and Siobhán, 2004], de France et al. [France et al., 2004], [Raghu et al., 2006], de Muller et al. [Muller, 2006] et de Del Fabro et al. [DelFabro et al., 2006].

Le processus de tissage proposé par AspeCiS nécessite de définir trois types de métamodèles. Le premier métamodèle pour modéliser les exigences de base, celles des EEs, le deuxième pour modéliser les exigences aspectelles, celles des EAs et le troisième pour modéliser les opérateurs de tissage.

Le métamodèle de tissage d'AspeCiS est une extension du métamodèle générique proposé par Didonet Del Fabro et al. dans [DelFabro et al., 2006], dont le contexte opérationnel est mentionné dans la figure 4.8. Il consiste à produire un modèle de tissage `WM` représentant le mapping entre deux modèles, un modèle de gauche `LeftMM` et un modèle de droite `RightMM`. Les métamodèles de ces modèles, y compris le modèle `WMM` (Weaving Meta Model), sont conformes au même métamétamodèle.

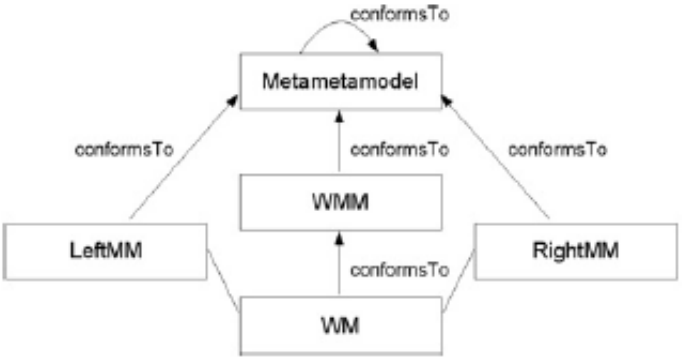


Figure 4.8 — AMW : Le métamodèle Atlas Model Weaver [DelFabro et al., 2006]

L'extension que nous proposons dans AspeCiS, du métamodèle de Didonet Del Fabro, consiste en la définition d'un métamodèle de base `ALeftmm` (pour : AspeCiS letf metamodel), un métamodèle d'Aspect `ARightmm` (pour : AspeCiS right metamodel) et un métamodèle de tissage qu'on appele `AWM` (pour : AspeCiS Weaving Metamodel), comme le montre la figure 4.9.

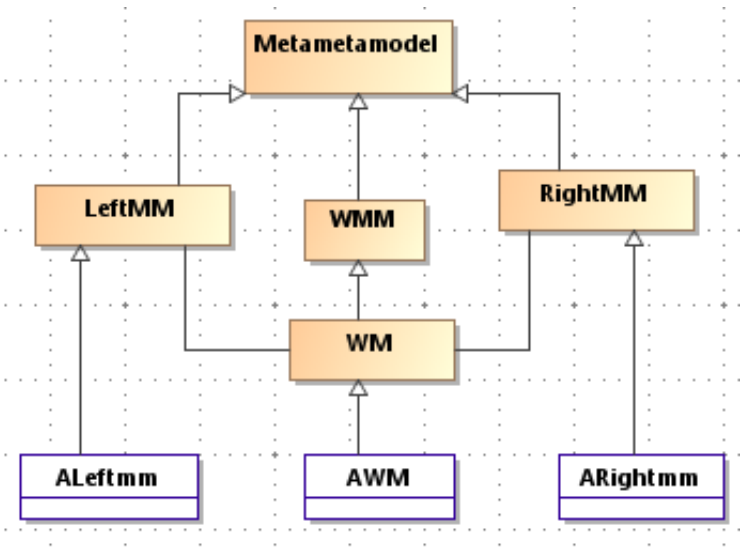


Figure 4.9 — AspeCiSWM : Le métamodèle de tissage d'AspeCiS

A cet effet, le métamodèle de base (`ALeftmm`) est utilisé pour modéliser les EEs, il est conforme au même métamodèle que celui d'UML. Le métamodèle aspect (`ARightmm`) est utilisé pour modéliser les EAs. Le métamodèle de tissage (`AWM`) est employé pour capturer les différents types de liens entre les éléments de deux premiers modèles (`ALeftmm`,

ARightmm). Il modélise les opérateurs de tissage. La section suivante présente en détail ces trois métamodèles.

4.6.1 Le métamodèle de base

Un système peut être décrit de manière structurelle ou comportementale. Le travail effectué dans cette thèse se focalise sur le développement du coté structurel du système à développer. A cet effet, nous utilisons le diagramme de classe d'UML pour représenter les EEs. La figure 4.10 décrit un métamodèle simplifié d'UML pour les diagrammes de classes qui représente le métamodèle de base d'AspeCiS (ALeftmm).

Une Classe (Class) contient des Attributs (Attribute) et des Methodes (Operation). Une Classe peut avoir des relations d'association avec d'autres classes. Une association (Association) est reliée à la Classe par le biais de AssociationEnd, elle peut être naviguable ou non. Les éléments de la classe (AssociationClass) héritent des deux éléments de (Class) et (Association) comme le montre la figure 4.10.

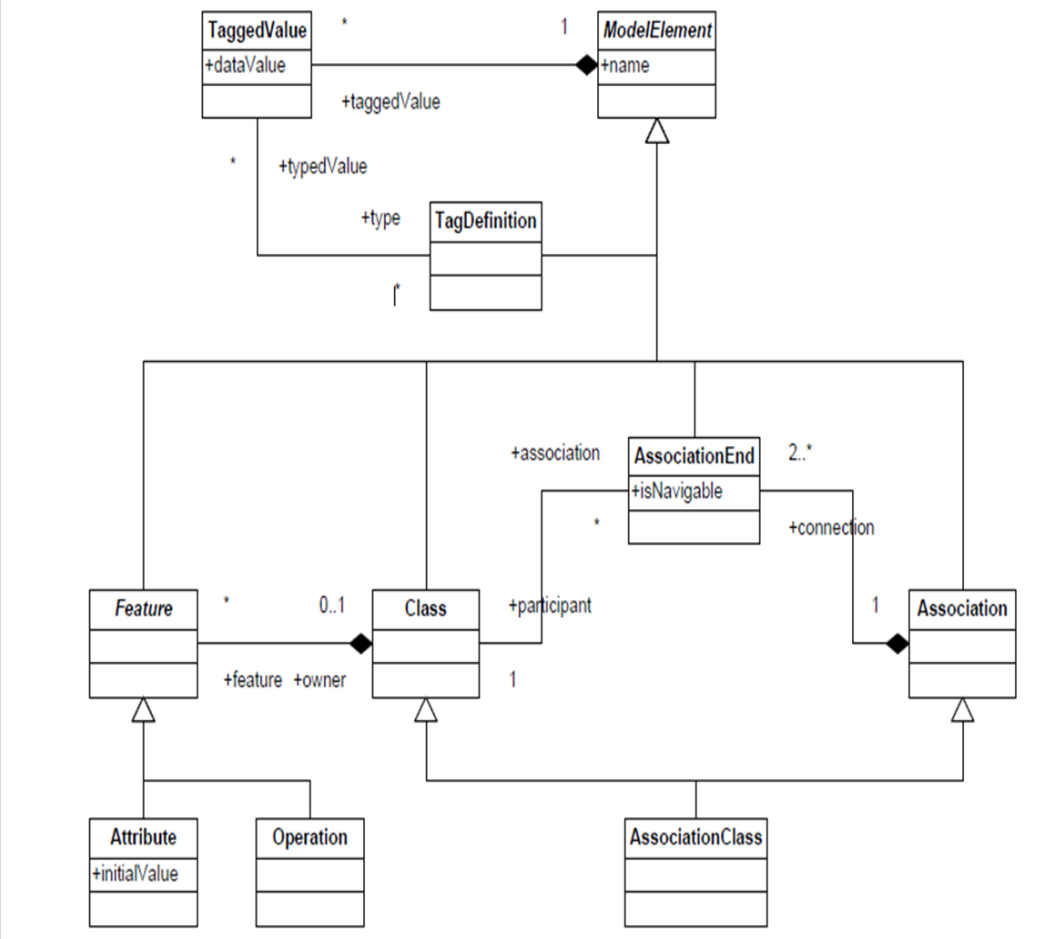


Figure 4.10 — Le métamodèle de base d'AspeCiS (ALeftmm)

4.6.2 Le métamodèle d'Aspect

Le `Rightmm` représente le métamodele d'Aspect nommé `Aspect`. L'`Aspect` peut être considéré comme une façon modulaire pour mettre en œuvre les préoccupations transversales que sont les exigences aspectuelles (EAs) [Awais et al., 2003]. Les aspects dans AspeCiS sont similaires aux classes. Cette similarité entre classe et aspect est mentionnée dans plusieurs travaux comme [Evermann, 2007], [Ubayashi et al., 2008], [Ubayashi et al., 2009], [Djabri and Amroune, 2012]. Cependant, l'`Aspect` contient des `Pointcuts` et des `Advices`. L'`Advice` définit le coté comportemental de l'`Aspect`. Quant aux `Pointcuts`, ils décrivent un ensemble d'emplacements dans le modèle de base dans lesquels interviennent les `Advices`. Les `Pointcuts` définissent le coté structurel de l'`Aspect`.

Le stéréotype `Aspect` étend la méta classe UML `Class`. Le stéréotype `Advice` étend la méta classe UML `BehavioralFeature`. Le stéréotype `Pointcut` étend la méta classe UML `StructuralFeature`. La Figure 4.11 illustre ces définitions.

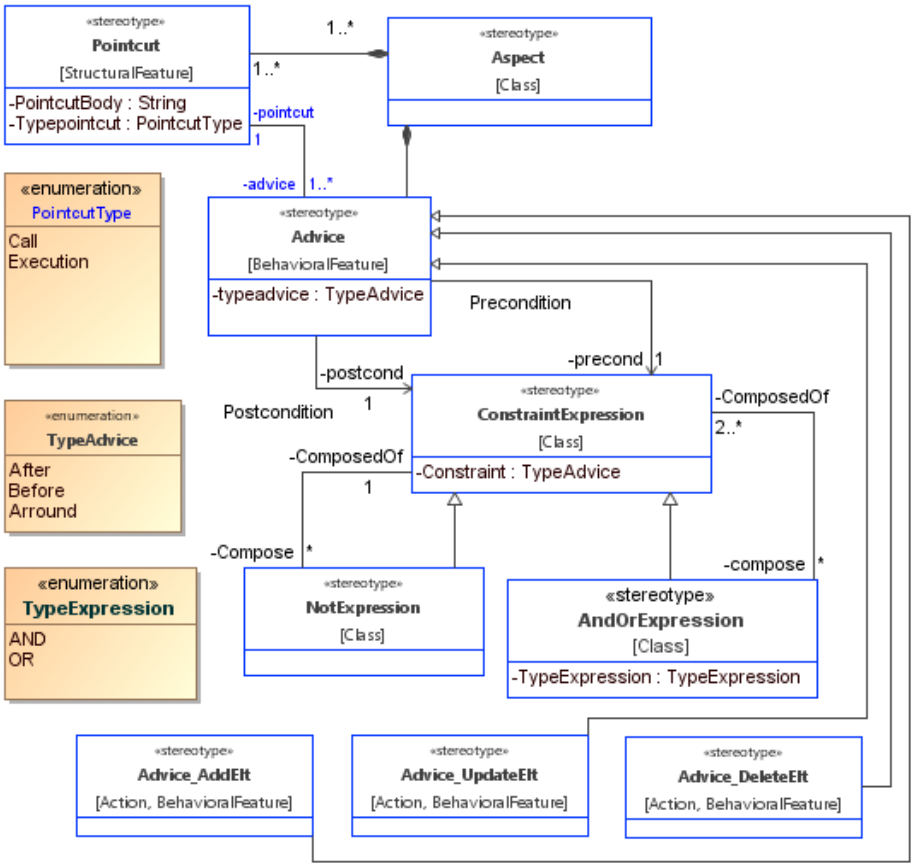


Figure 4.11 — Le métamodèle des Aspects (ARightmm)

Un `Advice` peut être soit un `Advice-AddElt` pour ajouter des éléments, ou bien un `Advice-Update` pour modifier des éléments ou encore un `Advice-DeleteElt` pour supprimer des éléments du modèle de base. Ces éléments peuvent être soit des attributs, des méthodes ou des associations. L'`Advice` peut être injecté avant (`Before`), ou après (`After`) un `Pointcut`. Le type `Arround` signifie que l'advice est exécuté à la place de `Pointcut` associé. Ces types sont déclarés dans la classe d'énumération `TypeAdvice`. Un `Advice` possède également, des préconditions et postconditions exprimant des contraintes.

Les contraintes permettent au concepteur de spécifier les prés-conditions à vérifier

avant l'exécution de l'`Advice`, et celles qui doivent être vérifiées après l'exécution (post-condition). Ces contraintes sont représentées par la classe `ConstraintExpression`. Elles sont exprimées en logique du premier ordre ayant un ensemble de paramètres combinés avec des opérateurs logiques.

A titre d'exemple, considérons une **EC** définissant une deuxième inscription d'un étudiant dans le même établissement universitaire. Pour qu'un étudiant puisse s'inscrire à une deuxième spécialité, il doit être déjà inscrit dans la première spécialité et la deuxième spécialité devra contenir des places pédagogiques disponibles. Si toutes ces contraintes sont satisfaites alors l'étudiant peut s'inscrire. Ces contraintes peuvent être exprimées en logique du premier ordre comme suit :

```
CR Second-Subscription (Sp : Specialty , St : Student)
Precondition : registered (St) and not full (Sp)
Postcondition : registered (St)
```

En d'autre terme, les préconditions déterminent les conditions qui doivent être vérifiées pour que l'`Advice` puisse s'exécuter. Alors que les postconditions sont les conditions qui doivent être vérifiées après avoir exécuté l'`Advice` (à cause de la relation entre la classe `Advice` et la classe `ConstraintExpression`).

4.6.3 Le métamodèle de tissage AWM

Le modèle de tissage `AWM` (cf. fig 4.12) a deux modèles en entrée (`ALeftmm` et `ARightmm`), qui sont tissés pour produire un modèle de sortie. AspeCiS utilise `AWM` comme un modèle de tissage pour capturer les différents types de liens entre les éléments des modèles d'entrée. Ces liens sont représentés par `AspeCiSWLink` qui a deux extrémités `EndBase` qui est un élément du modèle de base, et `EndAspet` qui représente un élément du modèle d'aspect. Les liens ont une sémantique différente, selon le scénario de l'application. Le lien (Attribut1, Classe1) est un exemple de `AspeCiSWLink` qui peut signifier que l'Attribut (Attribut1) du modèle `ALeftmm` doit être ajouté à la Classe (Classe1) du modèle `ARightmm`.

Le métamodèle de tissage d'AspeCiS est présenté dans la figure 4.12, dans lequel le modèle `Tissage-Base_Aspect` est une extension du `WModel`. Il est composé par deux modèles `Left` et `Right`. Il permet de tisser un modèle de base avec un ou plusieurs modèles d'aspect.

Le tissage des modèles (Base et Aspect) Le modèle `Tissage-Base_Aspect` est utilisé pour tisser des aspects (`Aspect`) avec des modèles de base des EEs (`Base`).

Le modèle de `Tissage-Base_Aspect` est composé de deux modèles (`Base` et `Aspect`), comme le montre l'extrait en langage KM3 ci-dessous. KM3 est un simple langage textuel pour définir des métamodeles [Grupe ATLAS, 2005].

```
class Tissage-Base_Aspect extends WModel
{
reference Base container : WModelRef;
reference Aspect
container : WModelRef;
}
```

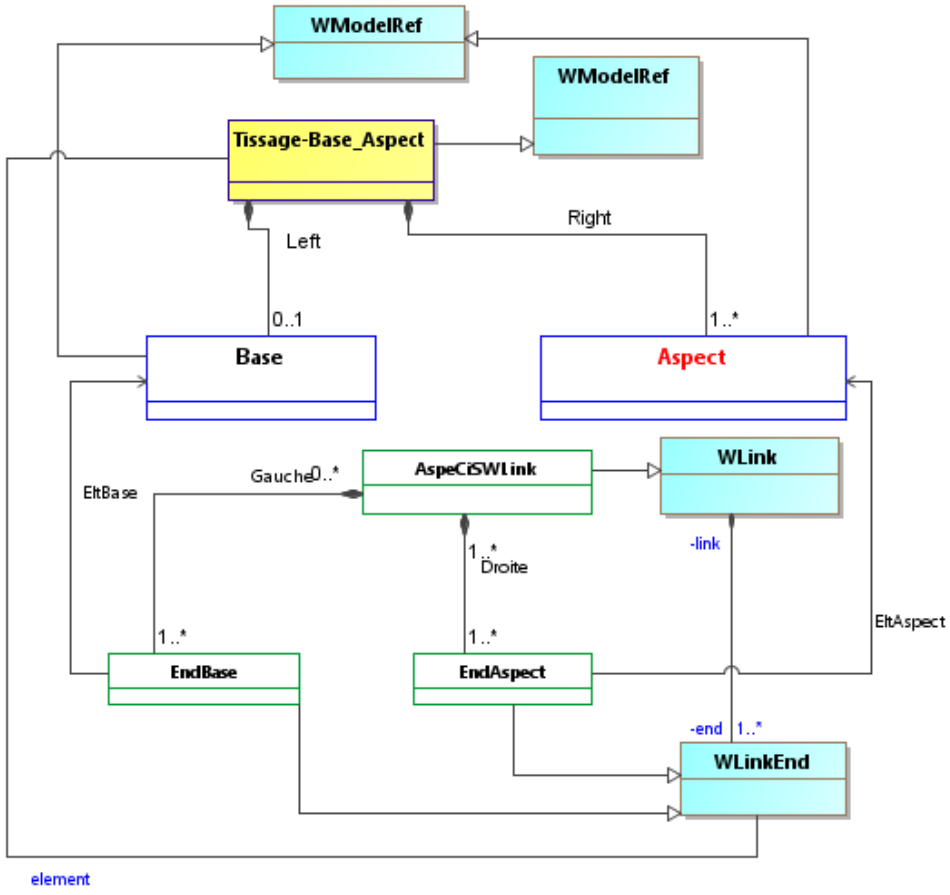


Figure 4.12 — Le métamodèle de tissage AWM

L'élément `AspeCiSWLink`, est une extension de `WLink`. Il contient deux éléments *Gauche* et *Droite* (`EndAspect`, `EndBase`), ces éléments étendent `WLinkEnd`. L'élement de gauche représente un artefact du modèle de `Base`, et celui de droite représente un

artéfact du modèle d'`Aspect`. Ces artefacts peuvent être des attributs des classes ou des associations.

4.7 Conclusion

Dans ce chapitre, nous avons présenté AspeCiS, l'approche d'élicitation, d'analyse et de modélisation des exigences que nous avons proposée pour un système d'information coopératif inter-organisationnel. AspeCiS réutilise des systèmes d'information existants pour développer un nouveau système d'information coopératif par une approche Aspect dirigée par les modèles.

Nous avons présenté les principes de base sur lesquels est construite AspeCiS : les exigences coopératives (ECs), existantes (EEs), supplémentaires (ESs) et aspectuelles (EAs), ainsi que le processus de tissage.

Ensuite, nous avons présenté l'approche AspeCiS. Cette dernière est composée de trois phases. La première phase a pour objectif l'elicitation et l'analyse des exigences. A ce niveau, un processus est défini par quatre sous phases qui sont la définition, le raffinement, l'expression des ECs en fonction des EEs et des ESs et la sélection des opérateurs de tissage. Nous avons présenté également un processus de résolution de conflits émergeant de la composition des aspects lors de la définition des ECs. La priorité des aspects en fonction des priorités des parties prenantes, le poids total et le nombre des parties prenantes, ainsi que la fréquence des aspects sont les critères de résolution de conflits utilisés par ce processus.

La deuxième phase d'AspeCiS montre bien comment sont modélisés les EEs et les ESs, les aspects et comment se fait le tissage de ces exigences avec les aspects. Pour cela, cette phase présente trois métamodèles pour modéliser les différentes catégories d'exigences .

Dans le prochain chapitre, nous appliquons AspeCiS à une étude de cas afin de montrer son utilisabilité.

Quatrième partie

Une étude de cas

5 Etude de cas

5.1 Introduction

L'objectif du présent chapitre est de présenter et discuter les principaux résultats de l'application de l'approche AspeCiS à un exemple de regroupement d'universités pour assurer une formation supérieure. Ce chapitre est organisé comme suit : nous commençons par introduire notre cas d'étude, et nous présentons le domaine d'application. Dans la section 2, nous appliquons l'approche AspeCiS à notre cas d'étude. Nous discutons les principaux résultats dans la section 3 et nous concluons dans la section 4.

5.2 Domaine d'application : la formation doctorale

Cette section commence par une brève introduction du domaine de la formation doctorale, notre domaine d'application avant de présenter le cas d'étude.

Dans le cadre de sa politique scientifique, le ministère Algérien de l'enseignement supérieur et de la recherche crée la formation doctorale organisée au sein des écoles doctorales.

L'école doctorale est la consécration d'un partenariat pédagogique et scientifique entre plusieurs établissements d'enseignement supérieur et repose dans son fonctionnement sur la ***coopération interuniversitaire*** autour d'objectifs communs préalablement définis. Outre la coopération inter-universitaire nationale l'école doctorale peut s'appuyer sur la coopération internationale. Cette coopération doit permettre l'appel à des compétences scientifiques pour la prise en charge de certaines fonctionnalités, l'organisation de séminaires ou de stages et la co-direction de thèses.

L'école doctorale consiste en une formation par la recherche, à la recherche et à l'innovation, qui peut être accomplie en formation initiale ou continue. Elle constitue une expérience professionnelle de recherche, sanctionnée, après soutenance de thèse, par

la collation du grade de docteur. L'école doctorale répond en priorité à des objectifs de mise en plan de formation des formateurs. Son objectif essentiel réside dans l'amélioration du rendement qualitatif et quantitatif du système national de formation supérieure de post-graduation par notamment la prise en charge du suivi permanent du post-graduant jusqu'à la soutenance de la thèse de doctorat.

L'école doctorale peut être domiciliée et organisée dans un établissement universitaire unique habilité pour cette formation et désigné point focal. L'école doctorale peut être aussi *organisée en réseau et domiciliée dans plusieurs établissements dûment habilités pour cette formation*. Dans ces deux cas, une convention de partenariat inter-universitaire doit préciser les formes de participation de chacun des établissements concernés par l'école doctorale (article 7 de l'arrêté n° 131 du 6 juin 2005).

Lorsque l'école doctorale est domiciliée dans un établissement de formation supérieure unique habilité, elle assure la formation d'étudiants issus également d'autres établissements universitaires avec la participation des enseignants qualifiés de ces établissements (extrait de l'article 8 du journal officiel). Lorsque l'école doctorale est organisée en un réseau de plusieurs établissements d'enseignement supérieur dûment habilités pour cette formation, elle est domiciliée dans chacun de ces établissements (article 9 du journal officiel).

5.3 Application de l'approche AspeCiS

Dans la section précédente, nous avons présenté les grandes lignes du domaine de la formation doctorale, d'où notre cas d'étude est issu. Dans cet exemple, nous nous intéressons particulièrement au cas où l'école doctorale est organisée en un réseau de plusieurs établissements d'enseignement supérieur dûment habilités pour cette formation et domiciliée dans chacun de ces établissements.

Comme nous l'avons montré et mentionné dans le chapitre 4, AspeCiS est structurée autour de trois phases. Dans cet exemple nous développons la phase 1, celle de l'élicitation et de l'analyse des exigences, ainsi que la seconde phase qui montre comment modéliser les exigences.

5.3.1 Phase 1 d'AspeCiS : élicitation et analyse des exigences coopératives

5.3.1.1 Définition des Exigences Coopératives

L'élicitation ou la découverte des exigences consiste en la collecte, la capture, des exigences coopératives (ECs) à partir d'une variété de sources, comme nous l'avons

mentionné dans la section 4.5. Après avoir utilisé et combiné plusieurs techniques d'éliciatation telles que des interviews suivies de séances de brainstorming regroupant les représentants de chaque université. La technique des cas d'utilisation est employée par l'équipe de l'IE pour compléter les ECs découvertes.

Le diagramme de la figure 5.1 présente un exemple de l'utilisation de la technique de cas d'utilisation employée par AspeCiS lors de la phase d'élicitation des exigences. Ce diagramme illustre les tâches du responsable de l'école doctorale.

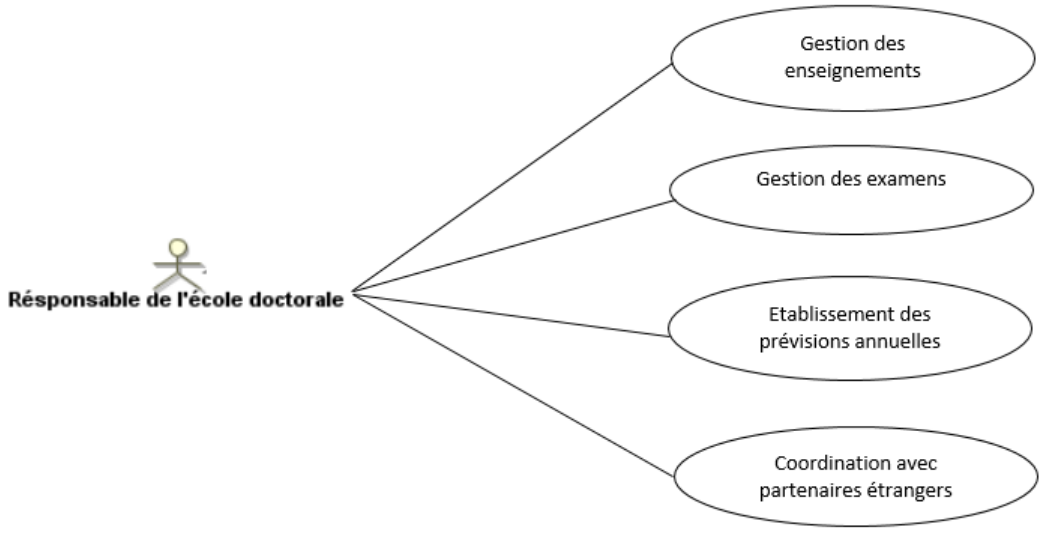


Figure 5.1 — Diagramme de cas d'utilisation de la mission du responsable de l'ED

Les ECs découvertes sont structurées dans le canevas du tableau 5.1. Ce canevas contient les champs suivants :

- *Code EC* : est un code structuré en 4 positions (a,b,c,d).
    1. La première position sert à indiquer le type de l'exigence, dans l'exemple (**EC**) pour désigner une Exigence Coopérative.
    2. La deuxième position indique le type du projet de coopération dans l'exemple (**ED.MI**) signifie qu'il s'agit d'un projet d'une école doctorale en maths informatique (MI).
    3. La troisième position précise l'année du projet coopératif, dans l'exemple (**12** pour l'an 2012)
    4. La quatrième position indique un numéro d'ordre séquentiel de l'EC dans le projet de coopération.
- *Libellé* : est le libellé de l'EC dans l'exemple *Concours d'accès.*

Code EC	Libellé	Description	Obs
EC-ED.MI-12-01	Concours d'accès	*Chaque établissement doit organiser le concours d'accès à l'ED*	*toutes les Universités*
EC-ED.MI-12-02	Modalité de participation	*Chaque établissement doit sélectionner les étudiants retenus pour passer le concours de l'ED selon les modalités de sélection*	*toutes les Universités*
EC-ED.MI-12-03	Gestion des enseignements	*Chaque établissement veille au déroulement des enseignement de l'ED*	*toutes les Universités*
EC-ED.MI-12-04	Gestion des examens	*Chaque établissement veille au déroulement des examens*	*toutes les Universités*
EC-ED.MI-12-05	Gestion des bilans	*Chaque établissement établit un bilan périodique du fonctionnement de l'ED*	*toutes les Universités*

Tableau 5.1 — Canevas des ECs

- *Description* : est une description textuelle de l'EC dans l'exemple de l'EC (**EC-ED.MI-12-01**), (`Chaque établissement doit organiser le concours d'accès à l'ED`)
- *Obs* : est une observation.

5.3.1.2 Raffinement des ECs

La phase de définition des ECs est suivie par une phase de raffinement. Cette dernière est composée de deux étapes. La première étape consiste en la décomposition des ECs et la deuxième est utilisée pour l'inférence.

La décomposition des ECs : elle consiste à décomposer les ECs en un ensemble d'exigences élémentaires. Le processus de la décomposition d'une EC génère un arbre à deux niveaux. Le premier niveau contient des des exigences supplémentaires (ESs) et/ou des exigences existantes (EEs). Les ESs sont décomposées à leur tour pour générer les exigences du dernier niveau de l'arbre qui sont soit des exigences existantes (EEs) soit des exigences aspectuelles (EAs). Les exigences (EEs & EAs) sont utilisées pour définir les ESs.

Pour illustrer cette décomposition, nous prenons comme exemple d'EC à décomposer, l'exigence **(EC-ED.MI-12-04)** relative à la *Gestion des examens* dans l'école doctorale

(cf. tableau 5.1). Le processus de décomposition donne l'arbre présenté dans la figure 5.2 où nous développons uniquement l'exigence *"Session Normale"* de l'exigence *Gestion des examens*.

Dans cet arbre, les exigences du premier niveau sont : *Préparation des sujets*, *Emploi du temps examen*, et la *Délibération*. Les exigences *(Notes examen)* et *(Règles admission)* sont des exemples des exigences du dernier niveau de l'exigence *Délibération* dans l'arbre de décomposition de la figure 5.2.

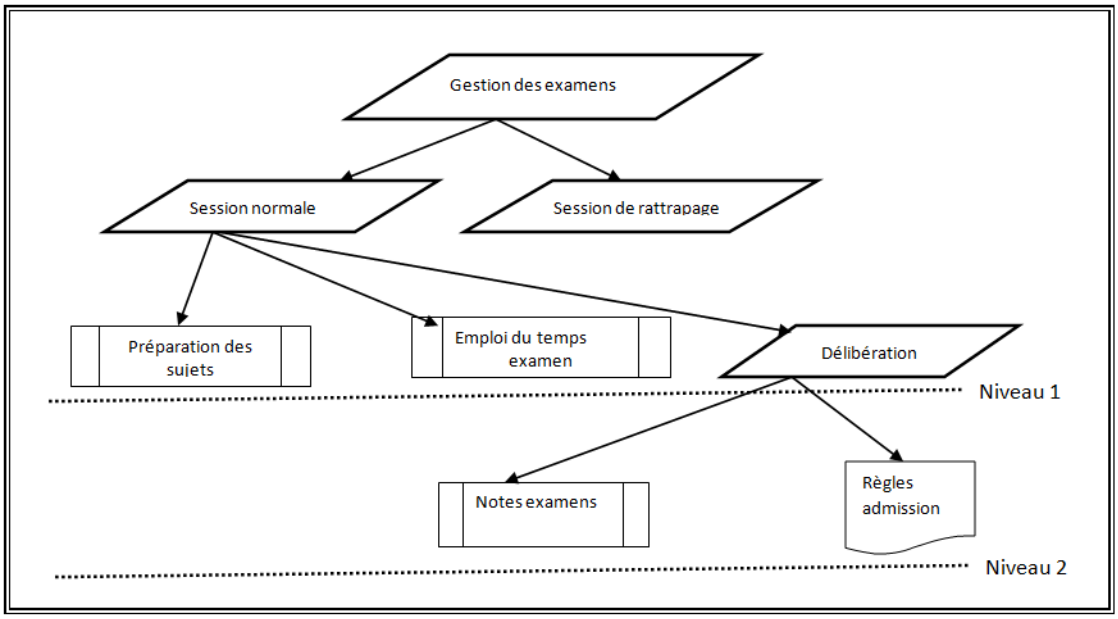


Figure 5.2 — Décomposition de l'EC *Gestion des examens* **(EC-ED.MI-12-04)**

Expression des ECs en fonction des EEs : après avoir décomposé l'EC en exigences élémentaires, cette sous phase définit les EEs et les EAs nécessaires pour la définition de l'EC. Dans l'exemple de la figure 5.3, l'ES *Règles admission* est le résultat du tissage de l'exigence EA1 avec les exigences (EE1, EE2, EE3).

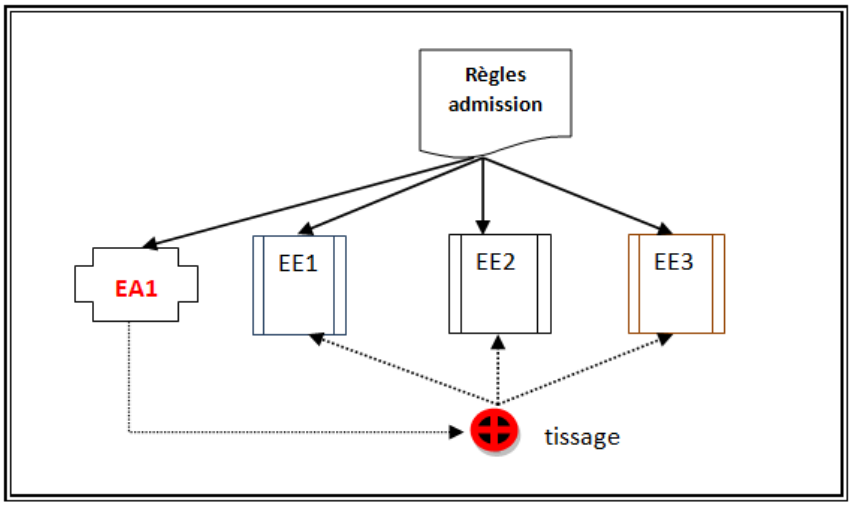


Figure 5.3 — Definition de l'ES **(Règles admission)** en fonction des EEs et EAs

Les exigences issues de la décomposition des ECs sont classées en deux grandes catégories celles des EEs et celles des EAs. A ce niveau, il est important de rappeler que les EEs peuvent être réutilisées sans aucune modification (comme les EEs du premier niveau de la figure 5.2), cependant, et c'est le cas général, ces EEs doivent être modifiées pour satisfaire les ECs. La modification de ces exigences fait émerger des exigences aspectuelles (EAs) qui seront tissées avec les EEs pour assurer les modifications appropriées de ces EEs. Le tissage des EAs est assuré par des opérateurs de tissage.

Dans l'exemple illustré dans la figure 5.2 (ou l'école doctorale se limite à trois universités) l'exigence coopérative *Délibération* est décomposée en deux exigences qui sont : l'exigence existante *Notes examen* et l'exigence supplémentaire *(Règles admission)*. Cette dernière définit les règles d'admission au niveau de l'école doctorale :

> L'ES *(Règles admission)* = *Les étudiants admis de chaque université doivent avoir plus de 70% des crédits des unités fondamentales* ***dont au moins 30% des crédit doivent être obtenus au premier semestre***.

Le processus de la décomposition de cette exigence illustré dans la figure 5.3 génère les exigences EEs (EE1, EE2, EE3). Ces exigences définissent les règles d'admission au niveau de chaque université :

> *EE1,EE2,EE3 = Les étudiants admis de chaque université doivent avoir plus de 70% des crédits des unités fondamentales.*

Pour être conforme avec les règles d'admission de l'ED, chaque université doit s'assurer que pour les 70% des crédits obtenus, au moins 30% de ces crédits doivent être obtenus en premier semestre. Pour satisfaire cette cette exigence, les universités de l'ED doivent modifier leurs règles d'admission pour pouvoir satisfaire l'EC des règles d'admission de l'ED. A cet effet, l'exigence (EA1) (*vérification des crédits d'admission)* est définie comme suit :

EA1=**plus de 30% des 70% des crédits sont obtenus au premier semestre.**

L'exigence EA1 est qualifiée d'exigence aspectuelle car elle doit être *tissée* avec les EEs existantes pour la définition de l'ES des *Règles d'admission* de l'école doctorale.

Jusqu'à présent, AspeCiS répond à la question **Quoi faire ?** (qui est exprimée par les ECs que le système à développer doit satisfaire), mais reste la question importante **Comment faire ?** c'est à dire comment arriver à définir ces exigences par le tissage des EAs avec les EEs. La section suivante apporte la réponse à cette question.

5.3.1.3 La sélection des opérateurs

A ce niveau, l'équipe de l'IE passe de la question *Quoi faire ?* à la question *Comment faire ?*. A ce niveau d'abstraction l'opération de tissage est attribuée à des opérateurs de tissage OP_T et de composition OP_C. Donc, l'opération de tissage de l'EA1 avec les EEs est définie par l'opérateur de tissage OP_T1. Son rôle est de tisser EA1 avec (EE1,EE2,EE3) afin d'apporter la modification adéquate aux EEs pour définir l'EC (*Règle admission*) de l'ED. En plus de l'opérateur de tissage OP_T1, un deuxième opérateur de composition OP_C1 est employé, son rôle est de composer les EEs (EE1, EE2, EE3) après les avoir modifiées par OP_T1. L'appel aux opérateurs de tissage et de composition pour définir l'EC (*Règle admission*) est défini de la manière suivante :

$$EC1 = Op\_T1(EE1, EA1) \tag{5.1}$$

$$EC2 = Op\_T1(EE2, EA1) \tag{5.2}$$

$$EC3 = Op\_T1(EE3, EA1) \tag{5.3}$$

$$EC = OP\_C1(EC1, EC2, EC3) \tag{5.4}$$

Pour résumer l'application de la première phase de l'approche AspeCiS, nous pouvons dire que pour l'exemple de l'exigence coopérative EC1 *Délibération* qui est décomposée en l'exigence existante *Notes examen* et l'exigence supplémentaire *Règle d'admission* définissant les modalités d'admission des étudiants en ED, cette exigence est définie par un processus de tissage (présenté dans les équations 5.1 à 5.4). Ce processus montre comment nous pouvons réutiliser des exigences existantes afin de définir une exigence

coopérative et ceci en tissant une exigence aspectuelle avec des exigences existantes. Le diagramme d'activité de la figure 5.4 illustre un exemple du processus de définition de l'exigence EC1. Une fois le processus de définition des ECs terminé, la deuxième

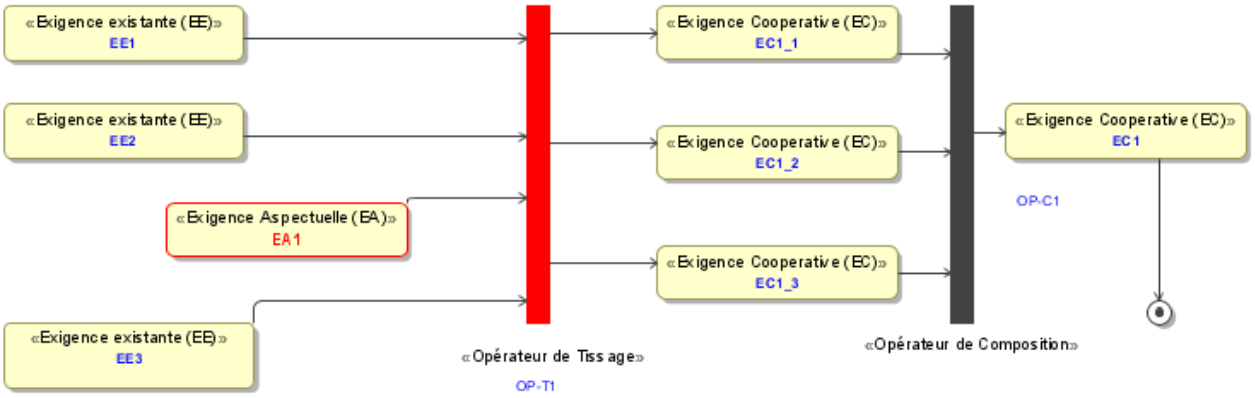


Figure 5.4 — Diagramme d'activité de définition de l'exigence l'EC1

phase d'AspeCiS consiste à définir les modèles des ECs obtenus aussi par un processus de tissage de modèles existants avec d'autres modèles aspectuels. La section suivante présente les détails de la phase II d'AspeCiS.

5.3.2 Phase II d'AspeCiS : modélisation des exigences coopératives

Il y a trois types de tissage comme le montre la figure 5.5. Le premier type de tissage est celui qui se fait au niveau des modèles (partie (a) de la figure 5.5), c'est à dire qu'une fois les deux modèles d'entrée tissés, le reste du processus est basé sur un seul modèle contenant les fonctionnalités de base tissées avec celles d'aspect et on reprend un cycle de développement classique où la séparation des préoccupations n'apparait pas. Le deuxième est un tissage au niveau du code (partie (b) figure 5.5), le principe de la séparation des préoccupations est maintenu jusqu'à la compilation du code, et le troisième type est un tissage au niveau exécution. Le premier type de tissage est utilisé par AspeCiS, car il cherche à utiliser le principe de la séparation des préoccupation dans les phases amonts de développement, en plus c'est un tissage facile à réaliser.

Cette section est consacrée à la modélisation de l'EC (EC1) relative aux *Règles d'admission*. La première phase d'AspeCiS nous a donné le processus décrivant la manière de définir cette EC en utilisant les EEs (EE1, EE2, EE3) et EA1.

Nous appelons M1, M2 et M3 les modèles des exigences EE1, EE2 et EE3 respectivement, et M4 le modèle de l'exigence EA1. Les modèles M1, M2 et M3 sont des

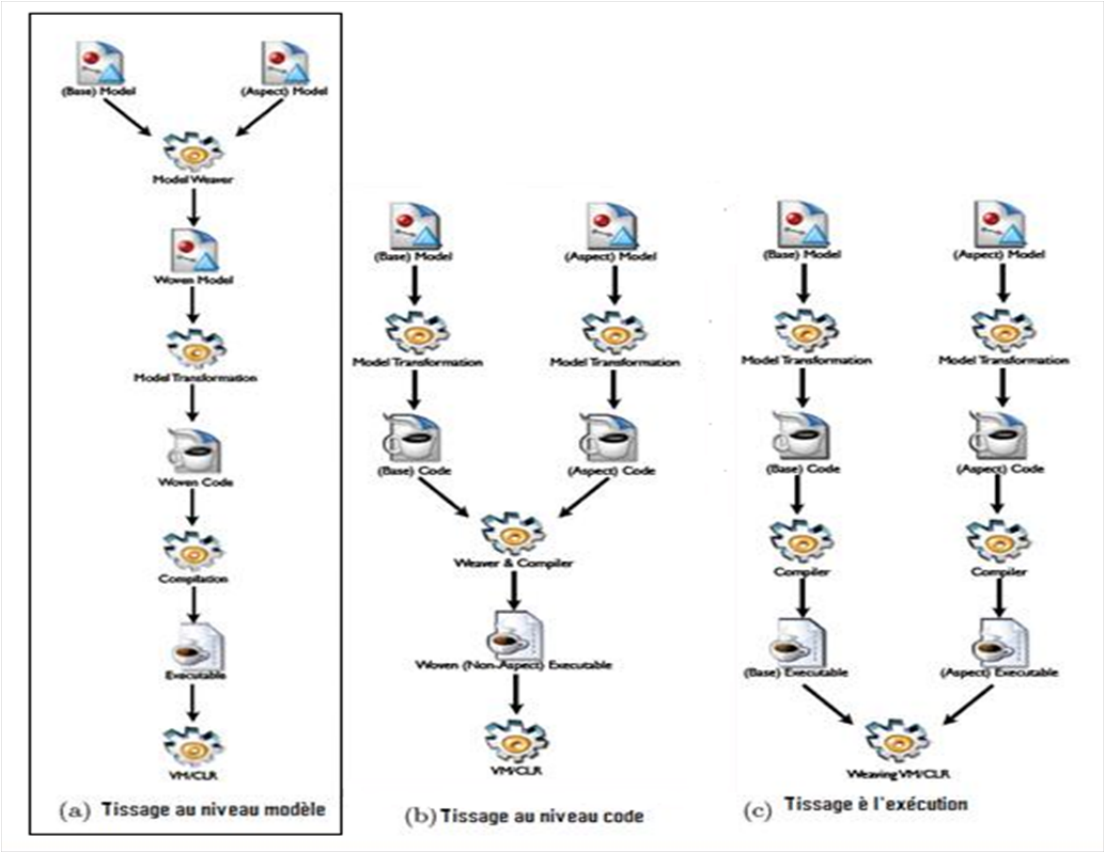


Figure 5.5 — Les types du tissage

diagrammes de classe conformes au métamodèle du diagramme de classe d'UML. Par contre M4 est un diagramme de classe mais qui est conforme au profil UML présenté dans le chapitre 4. Les sections suivantes présentent le modèle de base, d'aspect, le modèle de tissage et le modèle résultat pour l'étude de cas.

5.3.2.1 Le modèle de base (modèle des exigences existantes)

Cette section présente les modèles de base (M1, M2 et M3). Ces modèles sont identiques, car la politique d'admission dans notre exemple est supposée la même au niveau des trois universités de l'école doctorale.

Le diagramme de la figure 5.6 présente une partie du diagramme de classe des délibérations au niveau des trois universités. Ce dernier est composé de quatre classes (Etudiant, Université, Spécialité et Délibération).

Un étudiant est inscrit dans une école doctorale qui assure des spécialités. Au niveau de chaque spécialité, des délibérations sont effectuées. La méthode `effectueDélibération` de la classe `délibération` est utilisée pour cela. Lors des délibérations la règle d'admission au niveau de ces universités est celle des 70% des crédits obtenus. Cette méthode doit être modifiée afin de vérifier que dans les 70% des crédits obtenus, au moins 30% doivent être obtenus en premier semestre. A cet effet, l'appel de cette méthode constitue un point de coupure pour l'aspect assurant la modification adéquate pour l'admission des étudiants de l'école doctorale. La section suivante présente le modèle de cet aspect.

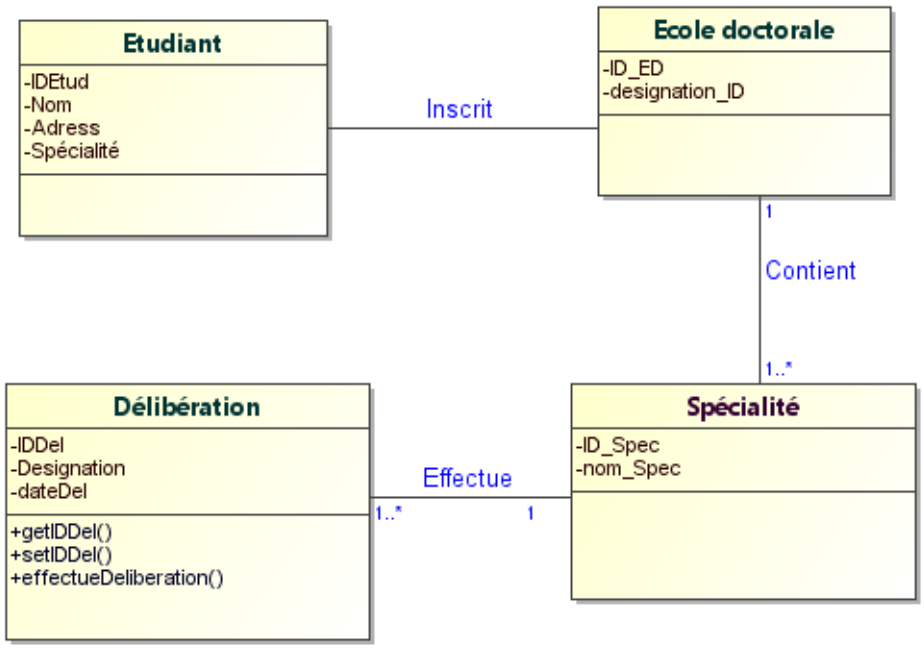


Figure 5.6 — M1 : Diagramme de classe représentant la délibération au niveau de chaque université

5.3.2.2 Le modèle d'Aspect

Le diagramme de la figure 5.7 représente un diagramme de classe du modèle d'exigence aspectuelle (ou modèle d'aspect) qui vérifie les règles d'admission au niveau de l'ED. Ce diagramme est conforme au profile UML d'AspeCiS. Il est composé d'une classe nommée `RègleAdmission` stéréotypé `Aspect` ayant un attribut `admission` stéréotypé `Pointcut` et d'une méthode stéréotypée `Advice_AddElt`. Cette méthode est appelée avant (`Before`) chaque appel de la méthode `effectueDélibération` de la classe `Délibération` des modèles de base (cf. figure 5.6).

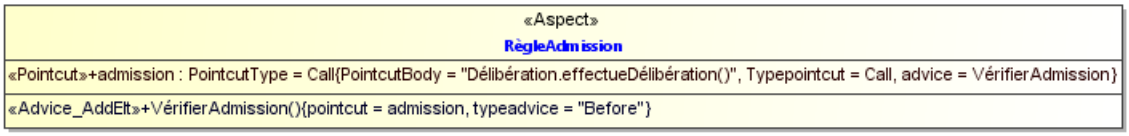


Figure 5.7 — M4 : Le diagramme de classe de l'Aspect "RègleAdmission"

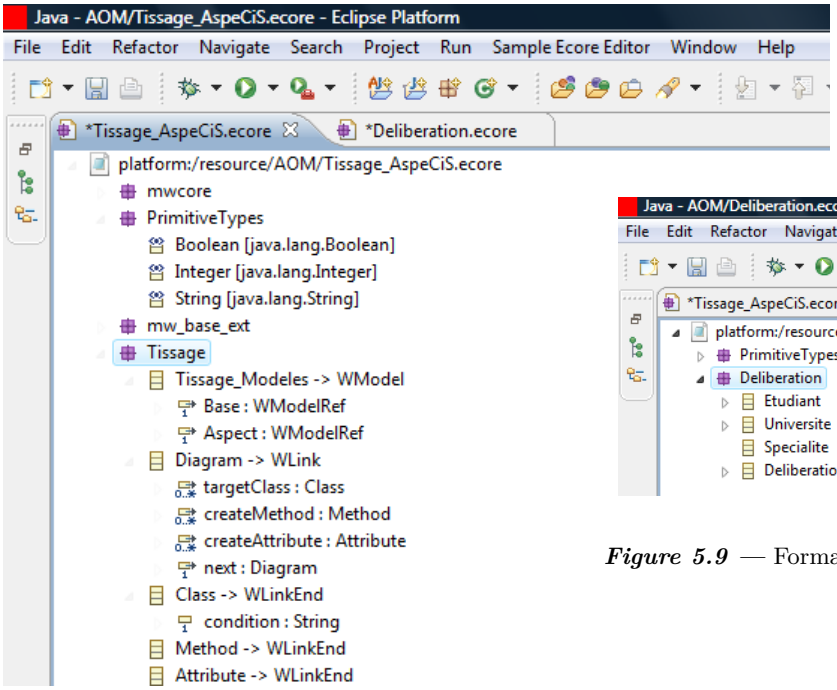
5.3.2.3 Implémentation du modèle de tissage AWM

Le modèle de tissage d'AspeCiS (AWM), qui représente à ce niveau les opérateurs de tissage définis dans la première phase d'AspeCiS, est utilisé pour tisser les modèles d'entrée (base et aspect). Ce modèle capture les différents types de liens entre les éléments des modèles d'entrée.

Pour implémenter et valider le processus de tissage proposé, nous avons utilisé la plateforme Eclipse avec les deux plugins EMF (Eclipse Modeling Framework) et AMW (Atlas model weaver) [DelFabro et al., 2006]. Le figure 5.8 présente le format ecore du modèle de tissage d'AspeCiS et les figures , 5.9 et 5.10 montrent le format ecore des modèles d'entrées (Base et Aspect). Le code suivant montre un extrait du modèle de tissage d'AspeCiS (AWM) exprimé en langage KM3.

```
package Tissage_AspeCiS {
class Weaving-Base_Aspect
extends WModel {
reference leftWCA container :Base;
reference rightWCA container:Aspect;}
class Base extends WModelRef {}
class Aspect extends WModelRef {}
class PointcutBaseAspect extends WLink {
reference leftModel
container : EndCore;
reference rightModel
container : EndAspect ;
reference PointcutCoreAspect : PointcutBaseAspect;}
```

Il contient `Weaving-Base_Aspect` comme un modèle de tissage qui est une extension de `WModel` du tisseur Atlas Model weaver (AMW). Le modèle `Weaving-Base_Aspect` contient deux modèles d'entrée (Base et Aspect). Dans cet exemple, nous utilisons `Weaving-Base_Aspect` pour ajouter une opération du modèle d'aspect qui est `AspectRegeleAdmission.VerifierAdmission` (cf. figure 5.7), au modèle de base qui est `Deliberation` (cf. figure 5.6), plus précisément à la classe `Deliberation`. Cette opération est appelée, avant `Before` l'appel de l'opération



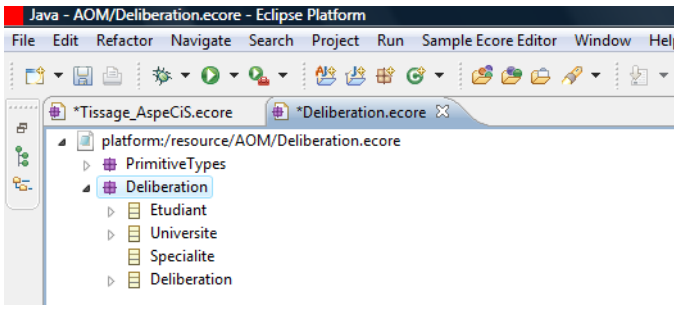


Figure 5.9 — Format ecore du modèle de base

Figure 5.8 — Format ecore du métamodèle d'AspeCiS

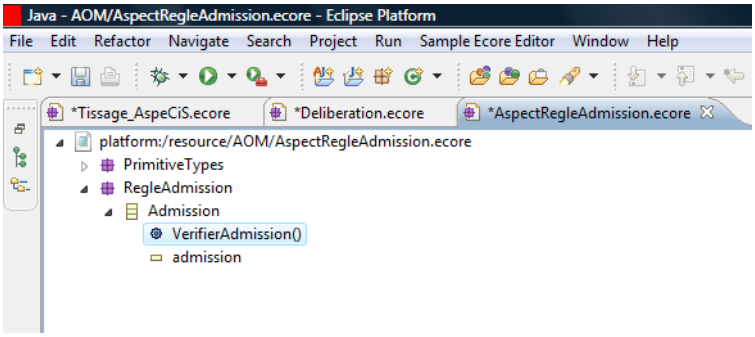


Figure 5.10 — Format ecore du modèle d'Aspect

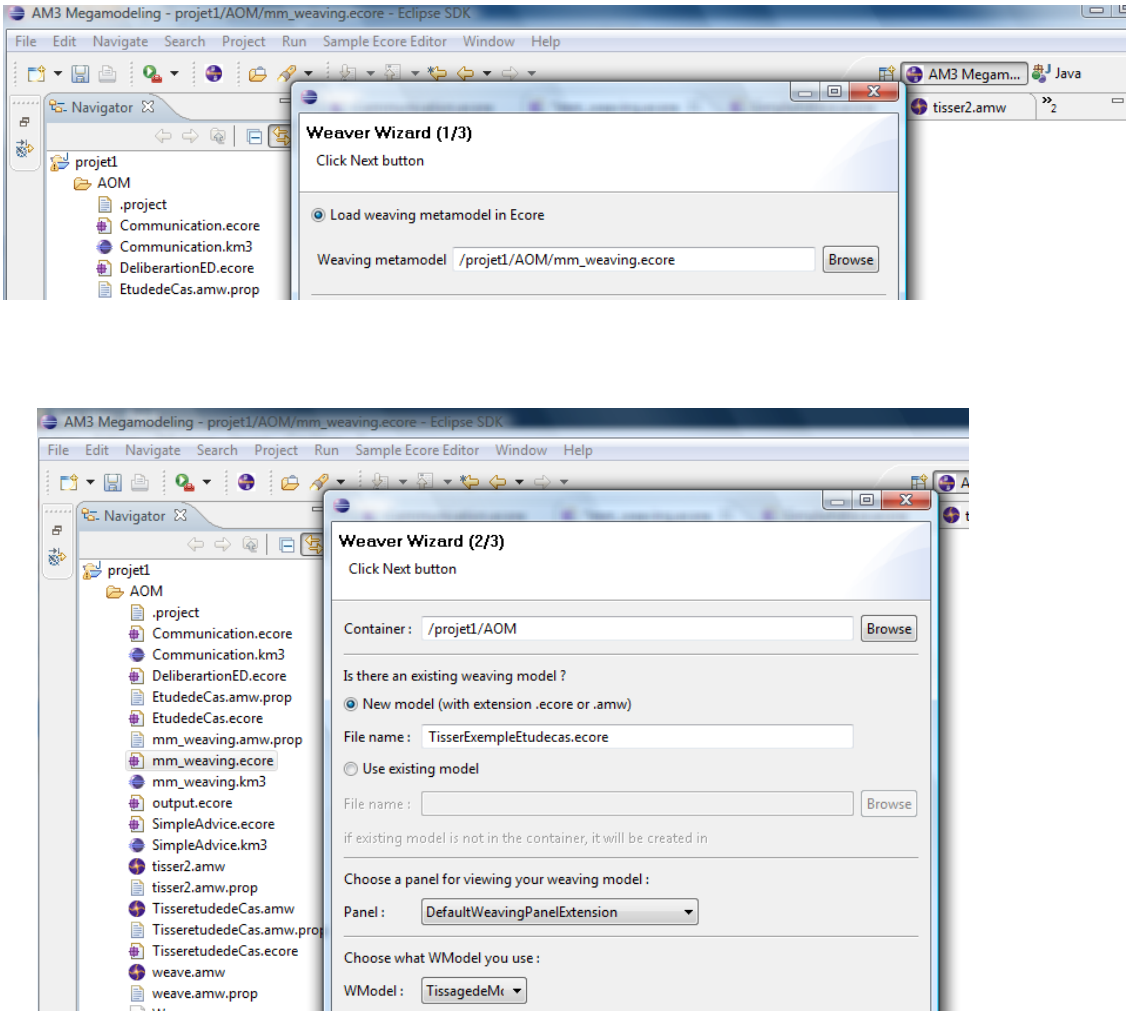


Figure 5.11 — La sélection du modèle de tissage

`Deliberation.effectuedeliberation()`, afin d'appliquer la règle d'admission des étudiants de l'école doctorale.

L'implémentation du modèle du tissage d'AspeCiS (AWM) sous éclipse, avec le plugin AMW, passe par trois étapes essentielles. La première étape consiste à définir les modèles de base et d'aspect en format ecore comme le montrent les figures 5.9 et 5.10, la deuxième

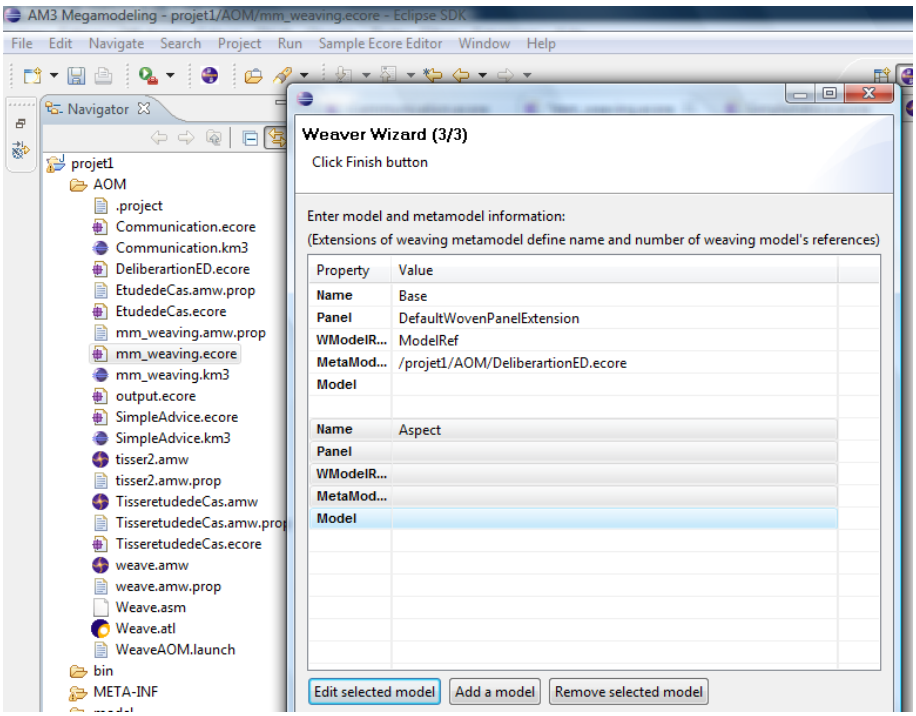


Figure 5.12 — La sélection du modèle de Base et du modèle d'aspect

étape consiste à définir le modèle de tissage d'AspeCiS en format ecore comme le montre la figure 5.8, quant à la troisième étape, elle consiste à appliquer AWM pour tisser les modèles d'entrées (Base et Aspect) en définissant les différents points de coupure (cf. panneau 2 de la figure 5.13).

La capture d'écran de la figure 5.13 résume le processus de tissage des modèles d'entrée. Elle est formée de trois panneaux. Le premier représente le modèle de base et le troisième représente le modèle d'aspect, quant au deuxième, il représente les points de coupure. Chaque point de coupure contient un élément de gauche qui est un artefact du modèle de base, et un élément droite représentant un artefact du modèle d'aspect. Pour notre exemple, le résultat du tissage du modèle de base et du modèle d'aspect donne un modèle de base modifié. Il contient tous les éléments du modèle de base augmenté d'une nouvelle méthode, celle qui vérifie les modalités d'admission en tenant compte des crédits obtenus par semestre. On peut dire que l'opération de tissage a ajouté les fonctionnalités manquantes pour réutiliser l'existant. La figure 5.14 représente le modèle résultat.

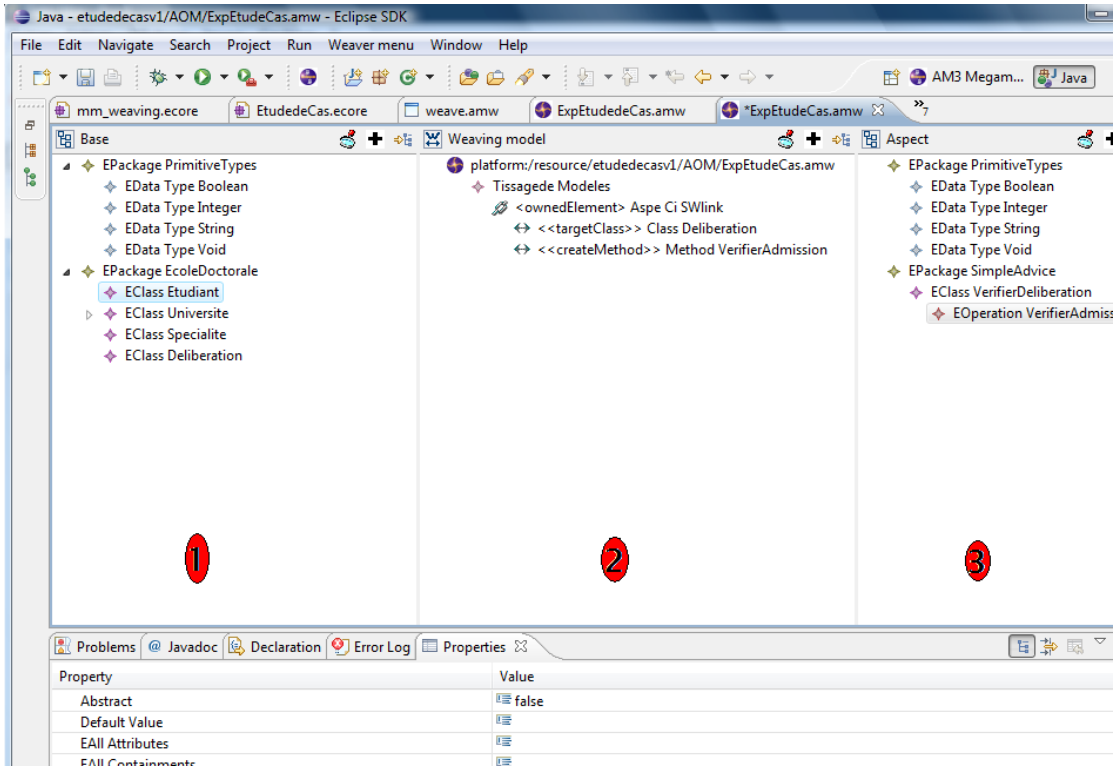


Figure 5.13 — La création des pointcuts (AspeCiSWLink))

5.4 Conclusion

Ce chapitre a présenté l'application de l'approche AspeCiS à une étude de cas issue du domaine de l'enseignement supérieur où un ensemble d'universités coopèrent pour assurer une formation supérieure dans une école doctorale. L'application a permis d'illustrer les étapes d'AspeCiS pour bien définir les exigences du futur système répondant aux exigences métiers exprimées dans le cahier des charges initial de la coopération.

Nous espérons dans l'avenir proche trouver un terrain d'expérience réel pour notre méthodologie qui permette de la valider dans la pratique. Cependant, l'application d'AspeCiS à cet exemple, a illustré, dans sa première phase, le processus de définition des exigences coopératives en réutilisant les exigences existantes conjointement avec des exigences aspectuelles émergentes de la définition des exigences coopératives du futur système à développer. Cet exemple a montré aussi l'utilisation d'un processus de tissage de modèle, présenté dans la deuxième phase d'AspeCiS, pour l'élaboration des modèle des exigences coopératives .

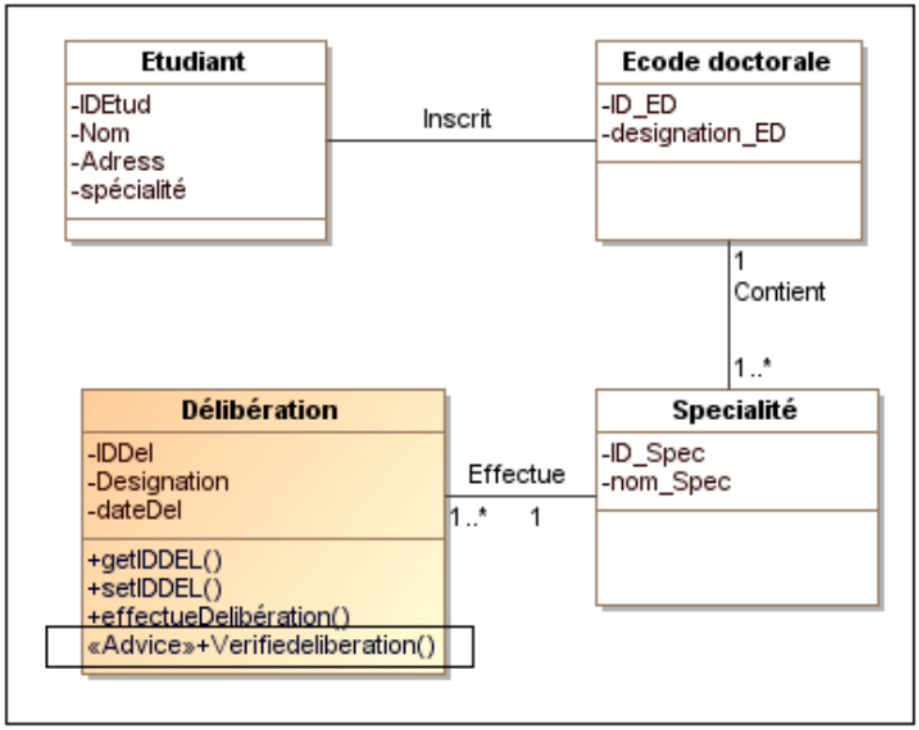


Figure 5.14 — Le modèle résultat du tissage

Il est également important de souligner l'utilité de gérer les conflits inter aspects qui peuvent apparaître lors de la phase de définition des exigences coopératives. Ce conflit est dû essentiellement aux influences négatives entre les aspects. La résolution des conflits dans AspeCiS est traitée par un algorithme de résolution de conflits basé sur la prioritisation des exigences et des parties prenantes.

Cinquième partie

Conclusion générale

6 Conclusion générale

La coopération inter entreprise est reconnue par les communautés scientifiques et professionnelles comme une forme d'organisation capable de prendre en charge des facteurs complexes émergeant de la mondialisation des marchés comme le raccourcissement du cycle de vie des produits, la concentration sur les métiers de base et le développement rapide des technologies de l'information et de la communication.

L'absence d'une démarche méthodologique rigoureuse de gestion des relations de coopération inter entreprise, conjointement avec l'environnement économique défavorable sont à l'origine de l'échec d'un certain nombre de coopérations. Du point de vue scientifique, le développement d'une telle démarche méthodologique est traité par la science d'ingénierie des systèmes.

Dans cette forme d'organisation, les systèmes d'information, capables de supporter la coopération inter entreprises sont considérés comme un élément central dans cette problématique. Des méthodologies, approches et outils de développement de ce genre de système d'information, dit système d'information coopératif (SIC), restent toujours un domaine d'actualité et de recherche.

Le choix du paradigme aspect dans un processus de développement d'un SIC trouve ses motivations dans l'apport principal apporté par ce paradigme dans le domaine de l'ingénierie des systèmes d'information. Ce paradigme permet au niveau implémentation de régler le problème de la dispersion et de l'enchevêtrement du code et au niveau conceptuel il permet une meilleure prise en charge des problèmes de préoccupations transversales.

En nous appuyant sur les concepts du paradigme aspect, qui permettent une meilleure réutilisation des artefacts des systèmes existants en tissant des aspects capables d'apporter les modifications adéquates aux éléments existants afin de les réutiliser, nous avons proposé une approche de développement d'un SIC, selon laquelle il est possible de construire un système à partir de systèmes existants produits à l'occasion de développements antérieurs. Dans cette approche nous avons abordé principalement deux phases

du cycle de développement logiciel à savoir l'élicitation et la modélisation des exigences.

6.1 Bilan des contributions

L'apport principal de notre travail est la proposition d'une approche baptisée AspeCiS qui propose une démarche de développement d'un SIC en développant deux activités importantes à savoir l'élicitation et la modélisation des exigences.

AspeCiS établit tout d'abord la description des catégories d'exigences exploitées, ainsi que le nouveau concept d'opérateur de tissage apporté par AspeCiS. Pour les types d'exigences, notre approche décrit les catégories des exigences suivantes :

- ***la catégorie d'exigence existante (EE)*** *qui représente un énoncé de service ou une contrainte fournis par un système existant.*
- ***la catégorie d'exigence aspectuelle (EA)*** *qui est un ensemble de comportements destinés à être tissés sur des EEs afin de les réutiliser.*
- ***la catégorie d'exigence supplémentaire (ES)*** *qui est définie comme étant une exigence qui ne peut pas être obtenue par une réutilisation simple des EEs. Elle est le résultat de la combinaison des* ***EEs*** *avec des* ***EAs****.*
- ***la catégorie d'exigence coopérative (EC)*** *qui est une exigence de but définie à partir des EEs et EAs. Elle présente quelles parties des exigences des SIs existants seront réutilisées et composées, et quelles parties doivent être nouvellement développées.*

Conjointement avec ces trois types d'exigences, AspeCiS définit aussi deux nouveaux concepts qui sont : les *Opérateurs de tissage* (OP-T) décrivant la manière de tisser des EAs avec des EEs, et les *Opérateurs de composition* (OP-C) décrivant la manière de composer des exigences pour définir des ECs. Toutefois, il est important de rappeler que la modification des EEs est considérée par AspeCiS comme le ***tissage*** d'un nouveau comportement décrit par l'EA sur les EEs. Un métamodèle d'EC est établi par AspeCiS pour définir toutes les catégories d'exigences et les relations existantes entre elles.

Lors de la phase d'élicitation, AspeCiS préconise un processus pour aider l'équipe d'IE, en particulier les analystes, à bien éliciter les exigences du SIC à développer. Ce processus est structuré autour de quatre phases qui sont :

- la définition des ECs du SIC : AspeCiS propose de combiner des techniques d'élicitation de groupe avec les techniques contextuelles, et éventuellement les techniques de la psychologie cognitive.
- le raffinement des ECs est utilisé pour éviter certains problèmes liés à la définition des ECs, tels que l'ambiguïté et l'incohérence.
- la décomposition des ECs est effectuée pour établir l'ensemble des EEs et EAs intervenant dans la définition des ECs. Ceci fait partie de la méthodologie d'analyse

proposée pour capturer les nouvelles exigences sous forme d'aspects.
– la définition des ECs en fonction des EEs et EAs : pour décrire la manière de réutiliser des EEs par le biais des opérateurs de tissage des EAs. Ceci constitue les points de jonction pour tisser des aspects fonctionnels sur les systèmes existants

Le tissage des EAs peut engendrer des conflits inter EAs. Pour cela, AspeCiS propose un processus de résolution des conflits basé essentiellement sur la priorité des acteurs impliqués dans la définition de chaque EC.

Dans sa deuxième phase, ASpeCiS développe le modèle des ECs selon le formalisme UML. Pour cela, elle propose un processus de tissage utilisant trois types de métamodèles. Le premier métamodèle pour modéliser des exigences de base, celles des EEs, le deuxième pour modéliser les exigences aspectuelles, celles des EAs et le troisième pour modéliser les opérateurs de tissage, celui du tissage proprement dit. Cette phase vise à répondre aux questions suivantes :
- pourra-t-on proposer des types d'aspects fonctionnels ? y retrouvera-t-on les modèles correspondants ?
? dans quelles conditions, un tissage fonctionnel est-il réapplicable/réutilisable ?

Pour montrer l'utilisabilité et la faisabilité d'AspeCiS, nous l'avons appliqué à un exemple du domaine de la coopération inter universités. Pour implémenter et valider le processus de tissage proposé, nous avons utilisé la plateforme Eclipse avec les deux plugins EMF (Eclipse Modeling Framework) et AMW (Atlas model weaver). Les différentes étapes de l'application de notre démarche sur cette étude de cas ont fait l'objet du cinquième chapitre.

6.2 Perspectives de recherche

La proposition de notre approche AspeCiS ne signifie pas qu'elle n'est pas susceptible d'être améliorée. Bien entendu, aucun système ne peut prétendre être parfait. D'abord le temps et la non disponibilité de plateforme d'expérimentation n'ont pas permis d'appliquer AspeCiS sur un vrai site réel. Ainsi afin d'améliorer AspeCiS et de la mieux évaluer, nous envisageons de l'appliquer dans un contexte réel.

La coopération inter-organisationnelle a un aspect international. Ainsi, lorsque les acteurs qui participent au processus d'élicitation sont de cultures différentes, ou même s'ils partagent une même langue maternelle, des significations différentes aux mêmes termes ou encore des termes différents ayant la même signification peuvent exister. De ce fait, le partage d'un vocabulaire commun du domaine d'application s'avère important. Le développement d'une *ontologie de domaine* peut servir comme solution à ce genre de problème. En plus, l'introduction de cette ontologie peut permettre ensuite la prise en

charge du coté comportemental du SIC, à développer par AspeCiS, qui reste un point important à développer pour donner plus d'efficacité et de complétude à AspeCiS.

Ce travail constitue une partie d'un projet de recherche qui vise à développer toutes les phases du processus d'ingénierie logicielle pour le développement d'un système d'information coopératif. Ainsi, une autre perspective aussi importante qui constitue la suite de notre travail de recherche, consiste à développer la phase d'implémentation proposée par AspeCiS dans sa démarche générale.

En conclusion, notre travail s'inscrit au sein d'un domaine pour lequel beaucoup de potentialités restent à découvrir et à exploiter. Nous avons proposé une méthodologie utilisant les apports importants du paradigme aspect, permettant l'élicitation et la modélisation des exigences pour un système d'information coopératif inter-organisationnel pour améliorer la flexibilité des systèmes d'information et, par là, la coopération entre organisations. Nous avons ouvert un certain nombre de perspectives dont le développement nécessitera un travail de longue haleine. Cependant, nous demeurons convaincus que ce travail doit être soutenu et poursuivi.

Bibliographie

AFIS. Ingénierie système, pourquoi ? comment ? In *Technical report, Association Française d'Ingénierie Système, http ://www.afis.fr/doc/presIS/presIS.htm*, 2007.

Mehmet Aksit. Separation and composition of concerns in the object-oriented model. *ACM Comput. Surv.*, 28(4es) :148, 1996.

Mehmet Aksit and Anand Tripathi. Data abstraction mechanisms in sina/st. In *OOPSLA*, pages 267–275, 1988.

Mehmet Aksit, Ken Wakita, Jan Bosch, Lodewijk Bergmans, and Akinori Yonezawa. Abstracting object interactions using composition filters. In *Proceedings of the Workshop on Object-Based Distributed Programming*, 1994.

Mohamed Amroune, Jean-Michel Inglebert, Nacereddine Zarour, and Pierre-Jean Charrel. Aspecis : An aspect-oriented approach to develop a cooperative information system. In *Model and Data Engineering - First International Conference, MEDI 2011, Óbidos, Portugal*, volume 6918 of *Lecture Notes in Computer Science*, pages 122–132. Springer, 2011.

Mohamed Amroune, Jean Michel Inglebert, Nacereddine Zarour, and Pierre Jean Charrel. Article : A conflict resolution process in aspecis approach. *International Journal of Computer Applications*, 44(10) :14–21, 2012.

Mohamed Amroune, Jean Pierre Charrel, Nacereddine Zarour, and Jean Michel Inglebert and. Composition of aspectual requirements : A multi-criteria process for conflict resolution. *Journal of Software Engineering*, 8(2) :75–88, 2014a.

Mohamed Amroune, Nacereddine Zarour, Med Ridda Laouar, SB Sean, and Hakim Bendjenna. A multi-criteria process to resolve conflict in the composition of aspectual requirements. *Human Systems Management*, 33(1) :27–34, 2014b.

Annie Anton. Goal based requirements analysis. In *Proceedings of the 2nd International Conference on Requirements Engineering ICRE96*, pages 136–144, 1996.

Annie Anton, Michael McCracken, and Colin Potts. Goal decomposition and scenario analysis in business process reengineering. *Proceedings of the 6th International Conference CAiSE94 on Advanced Information Systems Engineering, Utrecht, the Netherlands, Springer Verlag*, pages 94–104, 1994.

Adil Anwar. *Formalisation par une approche IDM de la composition de modèles dans le profil VUML.* Thèse de doctorat, Université de Toulouse-le-Mirail, Toulouse, France, décembre 2009.

Rashid Awais, Peter Sawyer, Ana Moreira, and João Araújo. Early aspects : A model for aspect-oriented requirements engineerin. In *RE*, pages 199–202, 2002.

Rashid Awais, Ana Moreira, and Joao Araujo. Modularisation and composition of aspectual requirements. In *2nd International Conference on Aspect Oriented Software Development (AOSD), Boston, USA*, 2003.

Elisa Baniassad and Clarke Siobhán. Finding aspects in requirements with theme/doc. In *Workshop on Early Aspects (held with AOSD 2004), Lancaster, UK*, 2004.

Daniel Bardou and Christophe Dony. Split objects : a disciplined use of delegation within objects. In *Proceedings of the 11th Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA'96), San Jose, California, USA*, pages 122–137, 1996.

Chester Irving Barnard. *The fonctions of the executive.* Boston : Harvard University Press, 1983.

Eduardo Barra, Gonzalo Génova, and Juan Llorens. An approach to aspect modeling with UML 2.0. In *Proceedings of UML-AOM04.*, 2004.

Mark Basch and Arturo Sanchez. Incorporating aspects into the UML. In *Proceedings of the 1st international conference on Aspect-oriented software development*, 2003.

Thomas E. Bell and T. A. Thayer. Software requirements : Are they really a problem ? In *ICSE*, pages 61–68, 1976.

Jean Bézivin and Olivier Gerbé. Towards a precise definition of the omg/mda framework. In *ASE*, pages 273–280, 2001.

Jean Bézivin, Salim Bouzitouna, Marcos Didonet DelFabro, Marie-Pierre Gervais, Frédéric Jouault, Dimitrios S. Kolovos, Ivan Kurtev, and Richard F. Paige. A canonical scheme for model composition. In *ECMDA-FA*, pages 346–360, 2006.

Grady Booch. Describing software design in (ada). *j-SIGPLAN*, 16(9) :42–47, 1981.

Jan Bosch, Clemens A. Szyperski, and Wolfgang Weck. Component-oriented programming. In *ECOOP Workshops*, pages 70–78, 2002.

Imed Boughzala and Jean-Louis Ermine. *Management des connaissances en entreprise*. Hermès Science, 2004.

Lotfi Bouzguenda. How to design a loose inter-organizational workflow ? an illustrative case study. In *DEXA*, pages 1–13, 2005.

Salim Bouzitouna and Marie-Pierre Gervais. Composition rules for pim reuse. In *2nd European Workshop on MDA with Emphasis on Methodologies and Transformations*, pages 36–43. Computing Laboratory, United Kingdom location : Canterbury, UK, 2004.

Gilad Bracha and William Cook. Mixin-based inheritance. In *OOPSLA/ECOOP*, pages 303–311, 1990.

Frederick Brooks. *The mythical man-month - essays on software engineering (2. ed.)*. Addison-Wesley, 1995. ISBN 978-0-201-83595-3.

Jean Bézivin. In search of a basic principle for model driven engineering. *NovaticaUpgrade*, V :21–24, 2004.

Jean Pierre Campagne and Olivier Sénéchal. *"Les nouvelles exigences de Coopération", Coopération et connaissance dans les systèmes industriels, sous la direction de Soënen R. et Perrin J.* Paris, Hermès, 2002.

Pierre Jean Charrel. The viewpoint paradigm : a semiotic based approach for the intelligibility of a cooperative designing process. *Australian journal of Information Systems*, 10(1), 2002.

Pierre Cointe. Metaclasses are first class : The objvlisp model. *SIGPLAN Not.*, 22 : 156–162, 1987. ISSN 0362-1340.

Thomas Cottenier, Aswin van den Berg, and Tzilla Elrad. Motorola weavr : Aspect and model-driven engineering. *Journal of Object Technology*, 6(7) :51–88, 2007.

Alan Mark Davis. *Just Enough Requirements Management : Where Software Development Meets Marketing*. Dorset House Publ, 2005.

Christophe Dejours. *Travail : usure mentale. De la psychopathologie à la psychodynamique du travail*. Nouvelle édition augmentée. Paris : Bayard, 1993.

Marcos Didonet DelFabro and Patrick Valduriez. Semi-automatic model integration using matching transformations and weaving models. In *SAC*, pages 963–970, 2007.

Marcos Didonet DelFabro, Bezivin J., Jouault F., Breton E., and Gueltas G. AMW : A generic Model Weaver. In *International Conference on Software Engineering Research and Practice (SERP05)*, 2005.

Marcos Didonet DelFabro, Jean Bézivin, and Patrick Valduriez. Weaving models with the eclipse amw plugin. In *Eclipse Modeling Symposium, Eclipse Summit Europe*, 2006.

Essame Didier. La méthode b et l'ingénierie système. réponse à un appel d'offre. technical report. In *Université de Nantes*, 2002.

Simon Dik. *The theory of functional grammar, part I : the structure of the clause. Functional Grammar Series.* Fories Publications, 1989.

Mustafa Redouane Djabri and Mohamed Amroune. A uml profile for aspectc++. In *Proceedings of International Conference on Information Technology and e-Services (ICITeS), 2012*, pages 1–6. Conference Publications IEEE, 2012.

Olfa Djebbi and Marie Pierre Gervais. Mda :vers l'industrialisation de la construction d'application réparties. In *Rapport interne DEA SIR. Lip6*, 2003/2004.

Marc Ehrig, Steffen Staab, and York Sure. Bootstrapping ontology alignment methods with apfel. In *Proceedings of ISWC*, pages 186–200. Springer, 2005.

Ramez Elmasri and Shankrant Navathe. *Fundamentals of Database Systems.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 3rd edition, 1999.

Thomas Erickson. Notes on design practice : Stories and prototypes as catalysts for communication. In *In Scenario-Based Design : Envisioning Work and Technology in System Development, Ed J.M. Carroll*, 1995.

Amitai Etzioni. *Modern organizations.* Prentice Hall, Englewood Cliffs, 1964.

Joerg Evermann. A meta-level specification and profile for aspectj in uml. In *Proceedings of the 10th international workshop on Aspect-oriented modeling*, AOM '07, pages 21–27, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-658-5.

Jean-Marie Favre, Jacky Establier, and Mireille Blay-Fornarino, editors. *L'ingénierie dirigée par les modèles : au-delà du MDA.* Hermes-Lavoisier, Cachan, France, 2006. ISBN 2-7462-1213-7.

Charles Fillmore. The case for case. In *Universals in linguistic theory, Holt, Rinehart and Winston (eds.), Bach & Harms Publishing Company*, pages 1–90, 1968.

Robert Filman, Tzilla Elrad, Clarke Siobhán, and Mehmet Aksit. *Aspect oriented software development.* Addison Wesley Professional, 2004. ISBN 0-321-21976-7.

William Frakes and Isoda Sadahiro. Success factors of systematic reuse. *IEEE Softw.*, 11(5) :14–19, September 1994. ISSN 0740-7459.

Robert France, Indrakshi Ray, Geri Georg, and Sudipto Ghosh. Aspect-oriented approach to early design modelling. *IEE Proceedings - Software*, 151(4) :173–186, 2004.

Robert France, Fleurey Franck, Reddy Raghu, Baudry Benoit, and Sudipto Ghosh. Providing support for model composition in metamodels. In *Proceedings of the 11th IEEE International Enterprise Distributed Object Computing Conference*, EDOC '07, pages 253–, Washington, DC, USA, 2007. IEEE Computer Society.

David Garlan. Software architecture : a roadmap. In *ICSE - Future of SE Track*, pages 91–101, 2000.

Martin Glinz. An integrated formal model of scenarios based on statecharts. *Lecture Notes in Computer Science*, pages 254–271, 1995.

Adele Goldberg and David Robson. *Smalltalk-80 : the language and its implementation.* Addison-Wesley series in computer science. Addison-Wesley, 1983. ISBN 9780201113716.

Roberto Gorrieri and Heike Wehrheim, editors. *Formal Methods for Open Object-Based Distributed Systems, 8th IFIP WG 6.1 International Conference, FMOODS 2006, Bologna, Italy, June 14-16, 2006, Proceedings*, volume 4037 of *Lecture Notes in Computer Science*, 2006.

Joseph Gradecki and Nicholas Lesiecki. *Mastering AspectJ : Aspect-Oriented Programming in Java.* John Wiley & Sons, Inc., New York, NY, USA, 2003. ISBN 0471431044.

Paul Grefen, Nikolay Mehandjiev, Giorgos Kouvas, Georg Weichhart, and Rik Eshuis. Dynamic business network process management in instant virtual enterprises. *Computers in Industry*, 60(2) :86–103, 2009.

Grupe ATLAS. *KM3 : Kernel MetaMetaModel – Manual v0.3*, 2005.

GTIE. Groupe de travail ingénierie des exigences. In *Ingénierie des Exigences*, 2000. URL `http://www.afis.fr/nav/gt/ie/ie.html`.

Ouafa Hachani. *Patrons de conception à base d'aspects pour l'ingénierie des systèmes d'information par réutilisation.* PhD thesis, de l'Universite Joseph Fourier, Grenoble I, 2006.

Abdelkader Hammami. *Modélisation technico-économique d'une chaîne logistique dans une entreprise réseau*. PhD thesis, Canada, 2003.

David Harel. Statecharts : a visual formalism for complex systems. *Sci. Computer Program, 8*, 1987.

William Harrison and Harold Ossher. Subject-oriented programming (a critique of pure objects). In *OOPSLA*, pages 411–428, 1993.

William Harrison, Harold Ossher, Randall Smith, and David Ungar. Subjectivity in object-oriented systems. *OOPS Messenger*, 5(4) :131–136, 1994.

Peter Haumer, Klaus Pohl, and Klaus Weidenhaupt. Requirements elicitation and validation with real world scenes. *IEEE Transactions on Software Engineering,Special Issue on Scenario Management*, 24(12), 1998.

C. H. Holbrook. A scenario - based methodology for conducting requirements elicitation. *ACM SIGSOFT, Software Engineering Notes*, 15(1) :95–104, 1990.

Andrew Jackson and Clarke Siobhán. Initial version of aspect-oriented design approach. In *Technical Report AOSD-Europe-TCD-7. AOSD - Europe*, 2006.

Ivar Jacobson, Magnus Christerson, Patrik Jonsson, and Gunnar Overgaard. *Object oriented software engineering : A use case driven approach*. ACM Press, 1992.

Praveen K. Jayaraman, Jon Whittle, Ahmed M. Elkhodary, and Hassan Gomaa. Model composition in product lines and feature interaction detection using critical pair analysis. In *MoDELS*, pages 151–165, 2007.

Bakker Jethro, Ut Bedir Tekinerdogan, Clarke Siobhán, and Tcd Andrew Jackson. Survey of analysis and design. *Integration The Vlsi Journal*, pages 1–259, 2005.

Jean-Marc Jezequel. *Object-oriented software engineering with Eiffel. Eiffel in practice*. Addison-Wesley series in computer science. Addison-Wesley, 1996. ISBN 9780201113716.

Li Jiang. A framework for the requirements engineering process development. In *Department of Electrical and Computer Engineering, Calgary, Alberta*, 2005.

Ivan Jureta. Engineering requirements for information systems using kaos and request frameworks. In *IMRU/FUNDP*, 2005.

Ivan Jureta, Alexander Borgida, Neil A. Ernst, and John Mylopoulos. Techne : Towards a new generation of requirements modeling languages with goals, preferences, and inconsistency handling. In *RE*, pages 115–124, 2010.

Lewis Johnson Kevin Benner, Martin Feather and Lorna Zorman. Utilizing scenarios in the software development process. In *Information System Development Process, Elsevier Science Publisher B.V. (North-Holland)*, pages 117–134, 1993.

Gregor Kiczales and Erik Hilsdale. Aspect oriented programming. In *ESEC / SIGSOFT FSE*, page 313, 2001.

Gregor Kiczales, Jim des Rivieres, and Daniel Gureasko Bobrow. *The Art of the Metaobject Protocol*. MIT Press, 1991.

Gregor Kiczales, John Lamping, and Anurag Mendhekar. What a metaobject protocol based compiler can do for lisp. Technical report, Xerox PARC, 1994.

Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Lopes, Jean marc Loingtier, and John Irwin. Aspect oriented programming. In *ECOOP*. SpringerVerlag, 1997.

Anneke Kleppe, Jos Warmer, and WBast. *MDA Explained : The Model Driven Architecture - Practice and Promise*. Herlos, 2003.

Gerald Kotonya and Ian Sommerville. *Requirements Engineering - Processes and Techniques*. John Wiley & Sons, 1998.

Bent Bruun Kristensen and Johnny Olsson. Roles patterns in analysis, design and implementation. In *OOIS*, 1996.

Charles Krueger. Software reuse. *ACM Comput. Surv.*, 24(2) :131–183, June 1992. ISSN 0360-0300.

Axel Van Lamsweerde. Requirements engineering in the year 2000 : A research perspective. In *Proceeding of 22nd International Conference on Software Engineering, Invited Paper, ACM Press*, 2000.

Axel Van Lamsweerde, Anne Dardenne, Bruno Delcourt, and F. Dubisy. The kaos project : Knowledge acquisition in automated specification of software. In *presented at American Association for Artificial Intelligence, Spring Symposium Series, Stanford University*, 1991.

Nadia Lehoux, Sophie D'Amours, and André Langevin. Dynamique des relations interentreprises : mécanismes, barrières et cas pratique. *Revue française de gestion industrielle*, 2008.

Karl Lieberherr, Ignacio Silva-Lepe, and Cun Xiao. Adaptive object-oriented programming using graph-based customization. *Commun. ACM*, 37(5) :94–101, 1994.

Karl Lieberherr, Doug Orleans, and Johan Ovlinger. Aspect-oriented programming with adaptive methods. *Commun. ACM*, 2001.

Cristina Videira Lopes and Karl Lieberherr. Generative patterns. In *the 8th European Conference on Object Oriented Programming (ECOOP'94) Workshop on Patterns*, 1994.

Pericles Loucopoulos and Vassilios Karakostas. *System requirements engineering*. International Software Engineering Series. McGraw-Hill, 1995. ISBN 9780077078430.

Jeff McAffer. Meta-level programming with coda. In *Proceedings of the 9th European Conference on Object-Oriented Programming*, ECOOP '95, pages 190–214. Springer-Verlag, 1995. ISBN 3-540-60160-0.

Sean McDirmid and Wilson Hsieh. Aspect-oriented programming with jiazzi. In *AOSD*, pages 70–79, 2003.

Nenad Medvidovic. A language and environment for architecture-based software development and evolution. In *Proceedings of the 1999 International Conference on Software Engineering*, pages 44–53, 1999.

Jamel Meslati. *MAGE : Une approche ontologique de l'évolution dans les sytèmes logiciels critique et embarqués*. PhD thesis, PhdThesis de l'Universite d'Annaba, Algérie, 2005.

Bertrand Meyer. Eiffel : Version 3 and beyond. In *Eiffel*, 1992.

Bertrand Meyer. *Object-Oriented Software Construction*. Prentice Hall, 1997. ISBN 978-0-13-629155-8.

Tatsubori Michiaki, Chiba Shigeru, Marc olivier Killijian, and Kozo Itano. Openjava : A class-based macro system for java. In *Reflection and Software Engineering*, pages 117–133. Springer-Verlag, 2000.

Ronald Mitchell, Bradley Agle, and Donna Wood. Toward a theory of stakeholder identification and salience : Defining the principle of who and what really counts. *Academy of Management Review*, 22(4) :853, 1997.

Sébastien Mosser, Franck Chauvel, Mireille Blay-Fornarino, and Michel Riveill. Web services composition : Mashups driven orchestration definition. In *CIMCA/IAWTIC/ISE*, pages 284–289, 2008.

Alexis Muller. *Construction de systèmes par application de modèles paramétrés*. Thèse de doctorat, Université de Lille 1, Lille, France, décembre 2006.

Alexis Muller, Olivier Caron, Berna Carré, and Gilles Vanwormhoudt. Réutilisation d'aspects fonctionnels : des vues aux composants. *L'OBJET*, 9(1-2) :241–255, 2003.

Alexis Muller, Olivier Caron, Bernard Carré, Gilles Vanwormhoudt, and Salim Bouzitouna. Ingénierie multi-modèles : Projection flexible d'assemblages de modèles. In *LMO*, pages 167–182, 2007.

Alex Norta and Paul W. P. J. Grefen. A framework for specifying sourcing collaborations. In *ECIS*, pages 626–638, 2006.

Bashar Nuseibeh and Steve M. Easterbrook. Requirements engineering : a roadmap. In *ICSE - Future of SE Track*, pages 35–46, 2000.

Bechar Nuseibeh, Jeff Kramer, and Anthony Finkelstein. A framework for expressing the relationships between multiple views in requirements specification. *In IEEE Transactions on Software Engineering,IEEE CS Press*, 20 :760–773, 1994.

Harold Ossher, Matthew Kaplan, William Harrison, Alexander Katz, and Vincent J. Kruskal. Subject-oriented composition rules. In *OOPSLA*, 1995.

Harold Ossher, Matthew Kaplan, Alexander Katz, William Harrison, and Vincent J. Kruskal. Specifying subject-oriented composition. *TAPOS*, 2(3) :179–202, 1996.

Jens Palsberg, Cun Xiao, and Karl Lieberherr. Efficient implementation of adaptive software. *ACM Trans. Program. Lang. Syst.*, 17(2) :264–292, 1995.

Jens Palsberg, Boaz Patt-Shamir, and Karl Lieberherr. A new approach to compiling adaptive programs. *Sci. Comput. Program.*, 29(3) :303–326, 1997.

David Lorge Parnas. On the criteria to be used in decomposing systems into modules. *Commun. ACM*, 15(12) :1053–1058, 1972.

Jeffrey S. Poulin. Populating software repositories : Incentives and domain-specific software. *Journal of Systems and Software*, 30(3) :187–199, 1995.

Reddy Raghu, Ghosh Sudipto, Robert France, Greg Straw, James Bieman, Nathan McEachen, Eunjee Song, and Geri Georg. Directives for composing aspect-oriented design class models. *Proceedings Aspect Oriented Modeling workshop held with MODELS/UML*, 3880 :75–105, 2006.

Pawlak Renaud, Retaillé Jean-Philippe, and Seinturier Lionel. *La programmation orientée aspect pour Java/J2EE*. Eyrolles, 2004.

Colette Rolland, Georges Grosz, and Régis Kla. Experience with goal-scenario coupling in requirements engineering. In *Proceedings of the IEEE International Symposium on Requirements Engineering, Limerick, Ireland*, 1999.

Douglas T. Ross and Kenneth Schoman. Structured analysis for requirements definition. *IEEE Trans. Software Eng.*, 3(1) :6–15, 1977.

Kenneth Rubin and Adele Golberg. Object behavior analysis. *Communications of the ACM, 35(9)*, pages 48–62, 1992.

Jean Louis Rullière and Andre Torre. Les formes de la coopération inter-entreprises. *Revue d'économie industrielle*, 1995.

Mehrdad Sabetzadeh and Steve Easterbrook. An algebraic framework for merging incomplete and inconsistent views. *Requirements Engineering, IEEE International Conference*, volume 0 :306–318, 2005.

Nassima Sadou, Dalila Tamzalit, and Mourad Oussalah. How to manage uniformly software architecture at different abstraction levels. In *ER*, pages 16–30, 2005.

Ed Seidewitz. What models mean. *IEEE Software*, 20(5) :26–32, 2003.

Chiba Shigeru. A metaobject protocol for c++. In *Proceedings of the tenth annual conference on Object-oriented programming systems, languages, and applications*, OOPSLA '95, pages 285–299. ACM, 1995. ISBN 0-89791-703-0.

Clarke Siobhán. Extending standard uml with model composition semantics. *Sci. Comput. Program.*, 44(1) :71–100, 2002.

Patrick Smacchia and Sébastien Vaucouleur. Dossier spécial : AOP, Intérêts et Usages. *Ressource électronique*, 2003.

Yannis Smaragdakis. Optimal trace reduction for lru-based simulations. Technical report, T. R, 1998.

Adel Smeda, Mourad Oussalah, and Tahar Khammaci. Madl : Meta architecture description language. In *SERA*, pages 152–159, 2005.

Richard Soley and the OMG Staff Strategy Group. Model-driven architecture. Technical report, disponible à l'adresse : http ://www.omg.org/presentation.html, 2000.

Ian Sommerville. *Software Engineering.* 6th edn, Addison-Wesley, USA, 2001.

Ian Sommerville and Peter Sawyer. *Requirement Engineering- A Good Practice Guide.* John Wiley and Sons, 1997.

Ian Sommerville, Peter Sawyer, and Stephen Viller. Viewpoints for requirements elicitation : a practical approach. In *Computing Department, Lancaster University, Lancaster, LA1 4YR, UK*, 1998.

Olaf Spinczyk, Andreas Gal, and Wolfgang Schröder-Preikschat. Aspectc++ : an aspect-oriented extension to the c++ programming language. In *Proceedings of the Fortieth International Conference on Tools Pacific : Objects for internet, mobile and embedded applications*, CRPIT '02, pages 53–60, 2002. ISBN 0-909925-88-7.

Standish-Group. Chaos standish group internal report, http ://www.standishgroup.com/chaos.html. In *Standish Group*, 2009.

Bjarne Stroustrup. *The C++ programming language (3. ed.)*. Addison-Wesley-Longman, 1997. ISBN 978-0-201-88954-3.

Kent Stuart. Model Driven Engineering. In *Proceedings of IFM 2002*, LNCS 2335, pages 286–298. Springer-Verlag, 2002.

Francis Tapon. A transaction cost analysis of innovation in the organization of pharmaceutical r & d. *Journal of Economic Behavior and Organization*, 12(9) :197–213, 1989.

Anand Tripathi, Eric Berge, and Mehmet Aksit. An implementation of the object-oriented concurrent programming language sina. *Softw., Pract. Exper.*, 19(3) :235–256, 1989.

Naoyasu Ubayashi, Genya Otsubo, Kazuhide Noda, Jun Yoshida, and Tetsuo Tamai. Aspectm : Uml-based extensible aom language. In *Proceedings of the 2008 23rd IEEE/ACM International Conference on Automated Software Engineering*, ASE '08, pages 501–502, 2008. ISBN 978-1-4244-2187-9.

Naoyasu Ubayashi, Genya Otsubo, Kazuhide Noda, and Jun Yoshida. An extensible aspect-oriented modeling environment. In *CAiSE*, pages 17–31, 2009.

Shtern Victor. On object-oriented approach to program modularization. In M. H. Hamza, editor, *IASTED Conf. on Software Engineering and Applications*, pages 230–235. IASTED/ACTA Press, 2004. ISBN 0-88986-425-X.

Jorge Villalobos. *Federation de composants : une architecture logicielle pour la composition par coordination.* PhD thesis, Grenoble, 2003.

Klaus Weidenhaupt, Pohl Klaus, Matthias Jarke, and Peter Haumer. Scenario usage in system development : a report on current practice. In *IEEE Software, March*, 1998.

Karl Wiegers. *Software Requirements.* Microsoft Press, 2nd edn edition, 2003.

Niklaus Wirth. *Programming in Modula-2.* Springer-Verlag, New York, NY, third edition, 1985.

David Wood, Michael Christel, and Scott Stevens. A multimedia approach to requirements capture and modelling. In *Proceedings. ICRE94, Colorado Springs*, 1994.

Eric Yu. Modelling strategic relationships for process reengineering. In *PhD thesis, university of Toronto Canada*, 1994.

Yijun Yu, Julio Cesar Sampaio do Prado Leite, and John Mylopoulos. From goals to aspects : Discovering aspects from requirements goal models. In *International Conference on Requirements Engineering, Kyoto, Japan*, 2004.

Ncereddine Zarour. *A Negotiation Framework For Organizational Information Systems*. Thèse de doctorat, Université Mentouri de Constantine, Constantine, Algérie, 2004.

Alanna Zito, Zinovy Diskin, and Juergen Dingel. Package merge in uml 2 : Practice vs. theory. In *9th International Conference on Model Driven Engineering Languages and Systems (MoDELS06)*, pages 185–199. Springer, 2006.